LOS TRUCOS DEL SPECTRUM

ANTONIO BELLIDO

Los trucos del SPECTRUM

1986

PARANINFO S.A.

MADRID

IMPRESO EN ESPAÑA
PRINTED IN SPAIN

ISBN: 84-283-1440-3

Depósito Legal: M-41.173-1985

PARANINFO S.A. Magallanes, 25 - 28015 MADRID (3-3551

ALCO, artes gráficas. Jaspe, 34 - 28026 MADRID

INDICE DE MATERIAS

Los trucos del Spectrum

En las páginas que siguen, resumo aquellas tretas, rutinas y manipulaciones que, de específica aplicación en el Spectrum, me han resultado útiles a lo largo de mi trabajo alrededor de esta máquina. Espero que su lectura contribuya a hacer las cosas más fáciles y divertidas al programador del Spectrum.

El libro está concebido como un manual de consulta al que remitirse cuando se quiera buscar algún tipo de aplicación que no está, fácilmente, al alcance de los usuarios del Spectrum, no obstante, y por lo inesperado de los resultados de los "trucos", las páginas que siguen serán interesantes para todo aficionado a la programación.

Cada "truco" va incluido en una ficha y cada ficha está dividida en dos partes: en la primera se expone claramente el *cómo* hacer funcionar el "truco" en cuestión y el efecto que se consigue al aplicarlo y en la segunda se explica, para aquel lector que desee profundizar en el conocimiento de su computador, el *porqué* del mismo.

Los títulos dados a cada ficha son suficientemente claros para saber qué se puede esperar de la lectura de cada una, no obstante, en aquellos "trucos" que, con pequeñas variaciones, producen diferentes efectos, se han añadido subtítulos o breve explicación por alternativa con el fin de poder recurrir a ellos con facilidad al presentarse la ocasión.

Normalmente, cada una de las *tretas* que se exponen a continuación ofrecen al programador ingenioso, gran cantidad de posibilidades en la resolución de problemas específicos.

En el casete que contiene los ejemplos, se han situado las rutinas —cuando existen— a partir de la línea 9990 para que sean de utilidad directa para el programador, sin otro requerimiento que borrar las líneas

inferiores del programa ejemplo, bien por tecleado directo de nuevas instrucciones, bien por un MERGE, cuidando de no introducir números de línea superiores a la 9990, con este objeto los listados de los ejemplos van numerados de 10 en 10.

Las fichas con asterisco junto al título, indican que la utilidad que figura en ellas está incluida en el casete que, opcionalmente, acompaña y complementa el libro.

Los caracteres que figuran junto al asterisco corresponden al nombre de grabación del programa.

Para el lector que sólo conozca la programación en BASIC, le sugiero una lectura en profundidad del Manual del Spectrum, y muy especialmente, en lo referente a las VARIABLES DEL SISTEMA, el JUEGO DE CARACTERES y, si desea conocer un poco mejor su máquina, le recomiendo el estudio de los libros: LOS COLORES Y LOS GRAFICOS EN EL SPECTRUM y SPECTRUM: CURSO DE INICIACION AL CODIGO MAQUINA también editados por PARANINFO.

Todo ello le ayudará a entender mejor las explicaciones que aquí se dan.

TABULACION DE CADENAS CON JUSTIFICACION A LA DERECHA	FICHA 1 * "tabulación"

Efecto: En la presentación de informes procedentes del computador, nos encontramos con la necesidad estética y funcional de encolumnar debidamente ciertas listas de números, ya que si no cuidamos este detalle nos encontraríamos con situaciones como ésta:

```
Camisas     1820
Hilo        2
Botones     72
```

Cuando lo adecuado sería:

```
Camisas     1820
Hilo           2
Botones       72
```

Cómo conseguirlo:

Mediante la siguiente instrucción:

```
PRINT TAB (c – LEN s$) ; s$
```

c representa la columna –de 1 a 32– a partir de la cual y hacia la izquierda debe aparecer la cadena y *s* es la cadena a imprimir.

El siguiente ejemplo servirá de modelo:

```
10 INPUT "artículo" ; a$
20 PRINT a$ ;
30 INPUT "precio?" ; s$
40 GOSUB 9990
```

```
  50  INPUT "otro artículo? (s/n)" ; o$
  60  IF o$ >= "s" THEN GO TO 10
  70  STOP
9990  PRINT TAB (22 - LEN s$) ; s$
9991  RETURN
```

Explicación: Todo el truco para conseguir éste efecto está en el contenido del paréntesis, ya que al restar de la columna *c* la longitud de la cadena a imprimir, medida en caracteres *–LEN s$–*, la impresión de la cadena s$ se producirá de forma tal que el último carácter de s$ se producirá en la columna *c – 1*.

CENTRADO DE TEXTOS	FICHA 2 *"Centrar"

Efecto: En ocasiones es conveniente centrar la impresión de textos de una forma simétrica con respecto al eje vertical de la misma, con el fin de obtener resultados de este tipo:

```
HOLA AMIGOS
EL MAYOR ESPECTACULO
DEL MUNDO
ESTA A PUNTO DE COMENZAR
```

Cómo conseguirlo: Por medio de la siguiente subrutina:

```
9989 STOP
9990 DEF FN T (t$) = 16 – (LEN t$) / 2
9992 PRINT TAB (FN T (A$)) ; A$
9994 RETURN
```

a la cual nos remitiremos a lo largo del programa sin más que igualar el texto a centrar en la varible A$.

Añada las siguientes líneas y pruebe.

```
10 LET A$ = "HOLA AMIGOS"
20 GOSUB 9990
30 LET A$ = "EL MAYOR ESPECTACULO"
40 GOSUB 9990
50 LET A$ = "DEL MUNDO"
60 GOSUB 9990
70 LET A$ = "ESTA A PUNTO DE COMENZAR"
80 GOSUB 9990
```

Explicación: En la línea número 9990, definimos una función que determina el número que corresponde a la columna de pantalla donde debe comenzar la impresión de la variable genérica t$.

En la 9992 pedimos la impresión de la variable alfanumérica A$ –que debe venir definida antes de recurrir a la subrutina– a partir de la columna que calcule la FN con el valor derivado de la longitud de A$.

RELLENADO DE FICHAS PREIMPRESAS EN PANTALLA	FICHA 3 * "fichas"

Efecto: En ocasiones, y con objeto de facilitar la entrada de datos, conviene presentar en pantalla el formato del impreso a rellenar de tal forma que el operador o teclista, no necesite otra cosa que escribir los recuadros que aparezcan en pantalla.

Supongamos que queremos introducir los apellidos, nombres y teléfonos de una serie de personas, para lo cual hemos hecho una ficha de este tipo:

```
APELLIDOS ■■■■■■■■■■■■■■■■■■■■■■■■■■■■
NOMBRE ■■■■■■■■■■■■■■■■ TI: ■■■■■■■■■■
```

De tal forma que los cuadrados negros que aparezcan en la ficha, se irán sustituyendo por los caracteres que componen cada item a medida que se tecleen.

Si un item es más corto que lo que determinan sus cuadrados negros, bastará con apretar ENTER para pasar al siguiente. Al llegar al límite de los caracteres que cada concepto tiene asignado, el cursor pasa automáticamente al siguiente.

En caso de error al teclear, borraremos por el prodecimiento usual (DELETE).

Cómo conseguirlo: Mediante la rutina que se expone a continuación:

```
10 PRINT AT 1, 0 ; "APELLIDOS" ; AT 3,0 ; "NOMBRE" ;
   AT 3, 21 ; "TL :"
20 DATA 1, 10, 22, 3, 7, 13, 3, 24, 7
30 LET S$ = "............................"
```

```
 40  LET lc = 0: LET n = 0: REM "n = número de items.
     lc = longitud acumulada, en caracteres.
     lm = longitud máxima por item. l = línea. c = columna"
 50  FOR X = 1 TO 3 : READ l, c, lm: PRINT AT l, c; S$ (1 TO lm):
     NEXT X
 60  RESTORE
 70  READ l, c, lm
 80  LET c0 = c
 90  LET I$ = INKEY $
100  IF I$ = "" THEN GO TO 90
110  IF CODE I$ <> 12 THEN GOTO 150
120  IF c > c0 THEN LET c = c – 1: PRINT AT l, c; " "
130  IF lc > 0 THEN LET lc = lc – 1
140  BEEP .2, –5 : GOTO 90
150  PRINT AT l, c; I$: BEEP .2, CODE I$/3
160  LET c = c + 1 : LET lc = lc + 1
170  IF lc = lm OR CODE I$ = 13 THEN BEEP .3,0: LET lc = 0: GOTO
     190
180  GOTO 90
190  LET n = n + 1: IF n <= 2 THEN GOTO 70
200  INPUT "otra ficha?" ; o$
210  IF o$ = "s" THEN CLS: RESTORE : GOTO 10
220  STOP
```

Al ejecutar este pequeño programa se consigue el efecto propuesto, bajo los condicionantes fijados.

El lector con algún conocimiento de programación podrá ampliar este ejemplo a la medida de sus necesidades, acoplándolo a su gusto en el contexto de un programa más amplio.

Explicación: Entre las líneas 10 y 50 se crea el formato de la ficha propiamente dicha.

En las líneas 120, 130 y 140 se determinan las condiciones de borrado para los caracteres equivocados.

Entre la 150 y la 190 se organiza la impresión de caracteres y el retorno del carro, cuando se aprieta la tecla ENTER.

CLARIFICANDO UN LISTADO GRACIAS A LOS CODIGOS DE CONTROL	FICHA 4

Efecto: Resaltar partes de un listado con el fin de poder localizarlas con mayor facilidad.

Supongamos que se ha admitido, al definir las variables numéricas en el listado, escribir su nombre con brillo y, las variables alfanuméricas, con parpadeo.

Cómo conseguirlo: Apoyándonos en los códigos de control y, en el caso propuesto más arriba, el proceso sería el siguiente:

En un imaginario programa, la línea 90 contiene la definición de dos variables: una numérica y otra alfanumérica. Sea, por ejemplo:

```
90 LET Z = 0 : LET A$ = "ESPERE"
```

Según lo establecido la Z deberá estar sobreiluminada y la A$ parpadeando, para conseguirlo, y una vez escrita la línea, situe el cursor entre LET y Z y siga estas indicaciones:

1°.– Pase al modo extendido.

2°.– Apriete la tecla donde figura el 9, con lo cual se sobreiluminará toda la línea a la derecha del cursor.

3°.– Desplace el cursor a la derecha de la Z.

4°.– Pase nuevamente al modo extendido.

5°.– Apriete la tecla donde figura el 8, con lo cual mantenemos sobreiluminada exclusivamente la Z.

6°.– LLeve el cursor a la izquierda de la A.

7°.– Pase al modo extendido.

8°.– Apriete CAPS SHIFT y 9, con lo cual parpadeará toda la línea a la derecha del cursor.

9°.– Desplace el cursor a la derecha de A$.

10°.– Pase al modo extendido.

11°.– Apriete CAPS SHIFT y 8, con lo que sólo parpadeará la A$.

De esta forma conseguiríamos destacar los dos tipos de variables a lo largo de todo un listado y, de igual forma, podríamos haber obtenido diferentes colores según nos interesara, como veremos un poco más adelante.

Al utilizar los códigos de control debemos ser conscientes de que consumen memoria y, por tanto, debemos ser cuidadosos, sobre todo en razón de que, debido a su invisibilidad, puede darse el caso de que haya ocultos más códigos de los estrictamente necesarios.

Para eliminar los códigos de control, hay que empezar por saber donde están situados, para lo cual debemos editar la línea que queremos investigar, supongamos que la 90 del ejemplo previo, y desplazar el cursor a la derecha, si existen códigos notaremos que el cursor se queda "enganchado" de forma que parece no avanzar, aunque, en realidad, está superando los códigos que allí están situados.

Una vez localizados, bastará que procedamos a su borrado por el procedimiento normal (DELETE), apareciendo un símbolo de interrogación por cada código borrado.

Pruébelo sobre la línea 90 de nuestro ejemplo.

LOS CODIGOS DE CONTROL Y LA IMPRESION EN PANTALLA	FICHA 5

Efecto: Producir impresiones en pantalla con color, brillo y/o parpadeo, según nuestro interés, mediante la introducción de códigos de control en las instrucciones oportunas.

Cómo conseguirlo: Supongamos que uno de los items de una ficha, preimpresa en pantalla, deseamos que salga con papel azul y tinta amarilla y que la línea que contiene la instrucción es la siguiente:

```
50 PRINT AT 13, 13; "CONCEPTO"
```

y que la cadena de caracteres *CONCEPTO* debe reunir los requisitos dichos.

Una vez escrita la línea, el proceso es el siguiente:

1°.– Situar el cursor entre las comillas y la C.

2°.– Pasar al modo extendido.

3°.– Apretar la tecla 1 (color azul), con lo cual toda la línea situada a la derecha del cursor toma el color azul de papel.

4°. – Pasar al modo extendido.

5°.– Apretar CAPS SHIFT y 6 (color amarillo), con lo cual toda la línea situada a la derecha del cursor toma el color amarillo de tinta.

6°.– Desplazar el cursor hasta el final de la cadena, entre la 0 y las comillas.

7°.– Pasar al modo extendido.

8°.– Apretar la tecla 7 (color blanco), con lo cual toda la línea situada a la derecha del cursor toma el color blanco de papel.

9°.– Pasar al modo extendido.

10°.– Apretar CAPS SHIFT y 0 (color negro), con lo cual toda la línea situada a la derecha del cursor toma el color negro de tinta.

Resumiendo, con los dos códigos iniciales hemos conseguido los colores de papel y tinta deseados y con los dos últimos hemos retornado a los colores iniciales.

De esta forma, al ejecutar el programa (RUN/ENTER), la palabra CONCEPTO aparecerá impresa con los atributos especificados mediante los códigos de control.

OCULTANDO PARTE DE UNA LINEA DE PROGRAMA	FICHA 6

Efecto: Hacer desaparecer una instrucción en una línea multisentencia, o la línea completa.

Cómo conseguirlo: Dando a la instrucción en cuestión el mismo color de tinta que actualmente tenga el papel, mediante los códigos de control.

Supongamos, en este sentido, que queremos dirigir la lectura de un programa desde la línea 100 a la 9990 sin que aparezca en el listado, y que los colores actuales de papel y tinta son blanco y negro respectivamente.

La línea del programa completa podría ser:

```
100 LET n=n+1 : GOTO 9990
```

y la zona de la misma a ocultar es toda la que está a partir de los dos puntos e incluidos éstos. El proceso a seguir, según lo visto en las fichas anteriores sería:

1°.– Situar el cursor a la izquierda de los dos puntos.

2°.– Pasar al modo extendido.

3°.– Pulsar CAPS SHIFT y 7, con lo que desaparecerá todo lo situado a continuación del cursor.

4°.– Desplazar el cursor hasta estar seguros de que hemos alcanzado el extemo de la línea.

5°.– Pasar al modo extendido.

6°.– Pulsar CAPS SHIFT y 0 para volver al color de papel general.

Los tres últimos pasos hay que darlos a ciegas.

Una ampliación de lo dicho aquí consiste en hacer desaparecer todo el listado a partir de una línea determinada, para verlo en acción supondremos el siguiente listado:

```
100 LET n = n + 1
110 STOP
120 DATA "MADRID", "LONDRES", "ROMA"
130 GOTO 10
```

Para ocultar las dos últimas líneas, editaremos la línea 110, pasaremos el cursor al final de la misma y, una vez allí y mediante los códigos de control, hacemos el color de tinta igual al del papel.

VENTAJAS E INCONVENIENTES DE LAS LINEAS MULTISENTENCIAS	FICHA 7

Efecto: Nada varía desde el punto de vista de la ejecución de un programa, al sustituir varias instrucciones contenidas en sucesivas líneas de programa por una sóla que las contenga a todas, separadas por el símbolo:

Esta forma de actuar proporciona, a cambio de ningún esfuerzo, un aceptable ahorro de memoria, aunque, si hacemos estas líneas excesivamente largas, se tornarán enojosas cuando necesitemos manipularlas.

Es importante recordar que, al colocar un IF/THEM en medio de una línea multisentencia y, si no se cumple la condición impuesta en la primera parte de esta instrucción, todo el resto de las instrucciones que la siguen en la línea no serán leidas.

Cómo conseguirlo: Un caso donde es muy recomendable proceder de ésta manera es en la definición de variables, por ejemplo, las líneas:

```
10 LET T = 3
20 U = PI/PI
30 LET D = U + U
40 LET Z = NOT U
```

pueden ser sustituidas por una sóla de ésta forma:

```
10 LET T = 3 : LET U = PI/PI : LET D = U + U : LET Z = NOT U
```

Explicación: El ahorro de memoria que se produce al utilizar líneas multisentencias procede de la forma en que el Spectrum controla cada línea del programa.

La explicación que sigue, no sólo clarificará el punto que estamos tratando ahora, si no que, además, será de suma utilidad para alguna de las fichas que siguen.

Vamos a estudiar como se almacena la primera línea de cualquier programa, sin microdrives en uso, para sacar las conclusiones que vienen al caso.

En primer lugar, la dirección de memoria donde comienza la zona de memoria dedicada al almacenamiento del programa en BASIC, está dada por la variable del sistema *PROG*, la cual es valorada mediante:

```
PEEK 23635 + 256 * PEEK 23636
```

y cuyo resultado es 23755.

En segundo lugar, admitamos que la primera línea de ese programa sea:

```
10  LET A$ = "A"
```

Si tecleamos ésta línea en el Spectrum y la introducimos en memoria mediante ENTER, quedaría almacenada de la siguiente forma:

En las posiciones 23755 y 23756, el número de línea (10).

En las posiciones 23757 y 23758, la longitud de la línea (8), en caracteres, incluyendo el código de fin de línea (ENTER) y excluyendo el número de línea y los bytes de este control. La palabra BASIC se considera un carácter a este efecto.

En la posición 23759, el código del comando BASIC (241).

En la posición 23760, el código del primer carácter (65).

En la posición 23761, el código del segundo carácter (36).

En la posición 23762, el código del tercer carácter (61).

En la posición 23763, el código del cuarto carácter (34).

En la posición 23764, el código del quinto carácter (65).

En la posición 23765, el código del sexto carácter (34).

En la posición 23766, el código de ENTER (fin de línea) (13).

En resumen, por cada línea que generemos vamos a consumir 2 *bytes* para controlar su número de línea, otros 2 *bytes* para la longitud de la misma, y uno más para ENTER, sin contar, claro está, el consumo de bytes que haga la propia instrucción.

De aquí podemos deducir que por cada línea evitada ahorraremos un mínimo de 4 bytes.

VENTAJAS E INCONVENIENTES DE INTRODUCIR VARIABLES DIRECTAMENTE	FICHA 8

Efecto: Mantener en memoria variables definidas en modo directo y, por tanto, que no figuren en el listado del programa, con lo cual se produce el ahorro de memoria que se deriva del hecho de no generar nuevas líneas de programa.

Cómo conseguirlo: En primer lugar debemos estar seguros de que las variables en cuestión no van a ser borradas por una instrucción CLEAR o RUN, esto quiere decir que, para ejecutar programas que contengan variables introducidas en modo directo –y cualquier programa las puede tener–, debemos utilizar un GOTO o salvarlo con las instrucción de autoejecución, *evitando el comando CLEAR* en todo el listado.

Esto puede ser muy útil, no sólo para ahorrar memoria, si no para confundir a posibles *investigadores* de nuestro programa. Y, como es lógico, la forma de conseguirlo es simplemente introducir las variables deseadas en forma directa sin número de línea, por ejemplo:

```
LET U=PI/PI : LET Z=NOT U : LET D=U+U
```

seguido de ENTER.

Una vez hecho esto, bastará un PRINT U, en modo directo, para comprobar que el Spectrum retiene en memoria, cualquiera de las variables –numéricas y alfanuméricas– así definidas.

Al proceder a la grabación del programa en casete o microcinta, también se grabarán las variables con el valor que, a la sazón, tengan.

Los inconvenientes se derivan de la necesidad que nos creamos, al tener que anotar las variables que introducimos de este modo, para poder, en cualquier momento, manipularlas o, en todo caso, evitar duplicarlas, con lo cual perderíamos la variable inicial y su función.

SUSTITUCION DE CADENAS Y NUMEROS POR VARIABLES	FICHA 9

Efecto: Ahorro de memoria y mayor comodidad para el programador, en función de que, una vez definida una variable y consumida, por tanto, la memoria necesaria, sólo será necesario recurrir al nombre de la variable en cuestión para, con un pequeño consumo de memoria adicional, tener a nuestra disposición aquello que la variable representa.

Cómo conseguirlo: Basta con definir las variables en cuestión y, si es posible, en modo directo –siguiendo un criterio de selección en función del reiterado uso que se haga de ellas–. Este sería el caso, muy usual, del 0 y el 1 necesarios para controles de brillo, parpadeo y sobreimpresión y, en otro orden de cosas, con frases del tipo: APRIETE UNA TECLA PARA CONTINUAR.

Para ello, definiríamos las variables por cualquiera de los procedimientos usuales (directo o programación), en los casos propuestos más arriba podríamos recurrir a:

```
LET U = PI/PI : LET Z = NOT U: LET A$ = "APRIETE UNA
TECLA PARA CONTINUAR"
```

Una vez hecho esto, sólo necesitamos recurrir a U y Z para 1 y 0 con lo cual ahorramos los *bytes* derivados de pasar a manejar un carácter (el de la variable) en lugar del número que requiere 7 bytes, ya que se guarda en la forma de coma flotante.

En el caso de la variable alfanumérica, el ahorro será tanto mayor, cuanto más caracteres formen la cadena.

CONTROL DE MEMORIA DURANTE LA PROGRAMACION	FICHA 10 * "memoria"

Efecto: Ayudar al programador, durante su trabajo, para conocer en que situación se encuentra la zona de memoria dedicada al BASIC y sus diferentes controles.

Cómo conseguirlo: Mediante la siguiente rutina:

```
9990 CLS: STOP
9991 :::::::::::::::::::::::::::
9992 PRINT "Disponible", PEEK 23730 + 256*PEEK 23731 – (PEEK
     23641 + 256*PEEK 23642)
9993 :::::::::::::::::::::::::::
9994 PRINT "Consumido prog." PEEK 23627 + 256*PEEK 23628 –
     (PEEK 23635 + 256*PEEK 23636)
9995 :::::::::::::::::::::::::::
9996 PRINT "Variables", PEEK 23641 + 256*PEEK 23642 – (PEEK
     23627 + 256*PEEK 23628) – 1
```

Este programa auxiliar, al cual se puede recurrir en cualquier momento con un GOTO 9991, ocupa 390 *bytes* innecesariamente, ya que, por claridad, se han consumido casi el doble de los necesarios. Para reducir la memoria utilizada bastaría con eliminar los separadores (:) y utilizar la función VAL con lo que los números de las direcciones de memoria serían tratados como cadenas.

Explicación: La zona de memoria que estamos manejando es dinámica, razón por la cual sus posiciones de comienzo varían, viéndonos obligados a determinar sus valores por el contenido que actualmente tengan las variables del sistema que necesitamos manejar. En este

caso son las siguientes por orden de su posición en el mapa de memoria:

VARIABLE	*FIJA*	*DIRECCION*
PROG	Comienzo programa en BASIC	23635
VARS	Comienzo zona variables	23627
E_LINE	Comienzo control manipulación	23641
RAMTOP	Ultimo *byte* disponible	23730

En función de esto, para determinar la memoria disponible, sólo necesitaremos restar de la dirección el último *byte* disponible (RAMTOP) la dirección que nos dé la variable E_LINE por la cual determinamos la dirección donde acaba la zona dedicada a las variables –y comienzo del control de manipulación–, esto no es exacto, pero si suficientemente aproximado.

Por similar razón, restando de la dirección de comienzo de la zona de variables (VARS), la dirección de comienzo de la zona dedicada al programa en BASIC (PROG), determinamos la memoria ocupada por el programa actualmente incluida la rutina anterior, la cuál se eliminará cuando no sea útil.

Un razonamiento parecido, nos llevaría a fijar el consumo de *bytes* que las variables, definidas en el programa BASIC que estemos desarrollando, estén requiriendo.

RUTINA PARA RENUMERAR UN LISTADO

FICHA 11

* "renumerar"

Efecto: Cambiar los números de línea actuales del programa que esté en memoria por otra secuencia que nos resulte más interesante, de acuerdo con el siguiente criterio:

1°.– Indicar al ordenador el número de la primera línea a cambiar, según el orden actual (li).

2°.– Indicar, a continuación y según el orden actual, el último número de línea que deseamos sea cambiado (lf).

3°.– Dar el nuevo número de la primera línea a cambiar (nl).

4°.– Dar la nueva cadencia –o paso– para renumerar (p).

La rutina que se da a continuación, está situada, según convinimos, a partir de la línea 9990 por lo cual, en el caso de querer mantener otra rutina de utilidad –tal como la de *control de memoria*– deberíamos alterar los números de línea de una u otra, antes de proceder a un MERGE.

Cómo conseguirlo: Con la siguiente rutina:

```
9990 STOP : INPUT "Línea inicial?" ; li´"Línea final?" ; lf´"Nueva
     línea inicial?" ; nl´"Paso?" ; p
9993 LET cB = PEEK 23635 + 256*PEEK 23636 : LET cV = PEEK
     23627 + 256*PEEK 23628
9994 LET lr = 256*PEEK cB + PEEK (cB + 1)
9995 IF cB >= cV OR lr > lf THEN PRINT "Renumerado!" : STOP
9997 IF lr >= li THEN POKE cB, INT (nl/256) : POKE cB + 1, nl –
     256* INT (nl/256) : LET nl = nl + p
9999 LET cB = cB + PEEK (cB + 2) + 256*PEEK (cB + 3) + 4 : GOTO
     9994
```

Explicación: En la línea 9990, determinamos las variables de acuerdo con el criterio expuesto en EFECTO.

En 9993, fijamos, inicialmente, la variable *cB* a la dirección de memoria de comienzo del programa en BASIC que, como hemos visto en una ficha previa, contiene —con el siguiente byte— el primer número de línea. En la misma línea, segunda instrucción, hacemos *cV* igual a la dirección de comienzo de la zona de variables o, lo que es igual, final del área de memoria dedicada al programa en BASIC.

En 9994, averiguamos el primer número de línea, dejándolo igual a la variable *lr*.

En 9995, colocamos los límites para que la renumeración finalice.

En 9997, alteramos el número de línea con las dos primeras instrucciones y, con la tercera, actualizamos la variable *nl* al sumarle el valor del paso *(p)*. El condicionante inicial analiza si la línea a renumerar comienza —o no— con la línea inicial.

Finalmente, en la línea 9999, localizamos la dirección donde está guardado el primer byte indicativo de la siguiente línea a renumerar:

cB = dirección que contiene el primer byte de la línea *ya* renumerada.

PEEK (cB + 2) + 256*PEEK (cB + 3) = longitud de la línea *ya* renumerada.

4 = 2 bytes para el número de línea *ya* renumerada + 2 bytes por la longitud de ésta misma línea.

AUMENTANDO EL CHASQUIDO DE LAS TECLAS	FICHA 12

Efecto: Agudizar el sonido que se produce al apretar las teclas, con objeto de afianzar la sensación de las pulsaciones sobre el teclado.

Cómo conseguirlo: Bien en modo directo, bien como una instrucción de programa, alterando el byte contenido en la dirección 23609 (variable del sistema PIP), de la siguiente forma:

```
POKE 23609, 60
```

Para volver al ruido estándar, coloque un 0.

Esta modificación es muy útil en aquéllas aplicaciones que exijan un uso continuado del teclado para introducir información: rellenado de fichas, programas de gestion, etc, para lograr este efecto debemos colocar la citada instrucción al comienzo del listado, de forma que se tenga la seguridad de que ha sido leida antes de la primera interrupción programada en el listado.

Aún se puede aumentar el chasquido –y el sonido emitido por el Spectrum en general– conectando el cable MIC al casete y al computador y apretando en aquél las teclas PAUSE, REC y PLAY.

MAYUSCULAS AUTOMATICAS	FICHA 13

Efecto: Garantizar que un programa, o parte de él, utiliza exlusivamente mayúsculas.

Cómo conseguirlo: Introduciendo en el lugar adecuado, la siguiente instrucción:

```
POKE 23658, 8
```

Con lo cual el Spectrum utilizará sólo caracteres en mayúsculas. Para volver a minúsculas, la instrucción es:

```
POKE 23658, 0
```

Esta instrucción es muy cómoda y de gran aplicación para garantizar que todos los *imputs* se producen con este tipo de letras y, especialmente en las decisiones IF/THEM.

Más adelante, cuando echemos un vistazo a las variables del sistema, estudiaremos la forma de pasar el cursor a otros modos.

PASANDO LISTADOS CON EL DOBLE DE VELOCIDAD

FICHA 14

Efecto: Eliminar un SCROLL? de cada dos. Cuando un listado de programa o de impresión en pantalla supera la capacidad, medida en líneas, de impresión de aquélla, se produce un mensaje de SCROLL?, si se aprieta cualquier tecla que no sea la N o BREAK, otra pantalla se llena con la continuación del listado en cuestión y, así, el proceso se repite, una y otra vez, hasta concluir con la última línea a imprimir.

Cómo conseguirlo: Cuando aparezca el mensaje de SCROLL?, apretar simultáneamente CAPS SHIFT y la tecla 3 ó la tecla 4.

Este "truco" nos permite una aproximación más rápida, pero controlada, a una línea de programa o a un item de un listado producido por un programa y combinado con lo expuesto en la ficha 4, nos permite movernos a través de los listados con mucha comodidad y precisión.

En general una hábil combinación de todas las fichas, permite una programación más profesional, tanto en el desarrollo como en los resultados.

AVERIGUANDO LO QUE HAY IMPRESO EN PANTALLA, SE VEA O NO	FICHA 15

Efecto: Determinar el carácter que existe en cualquier posición de carácter en pantalla.

Cómo conseguirlo: Mediante la función SCREEN $ seguida de la línea y columna, entre paréntesis, de la posición de carácter a investigar.

Supongamos que en la posición 0, 0 hay una *a* impresa de forma que no sea visible gracias a los códigos de control, como ya vimos:

```
10 PRINT AT " "
20 PRINT SCREEN$ (0, 0)
```

al correr este programa, y en el supuesto de haber colocado una *a* invisible en la línea 10, aparecerá una *a* al comienzo de la segunda línea de caracteres como consecuencia de la función SCREEN$ de la línea 20.

BORRANDO PUNTOS EN LA PANTALLA

FICHA 16

Efecto: Hacer desaparecer un punto, impreso previamente, de la pantalla.

Cómo conseguirlo: Mediante PLOT INVERSE 1 seguido de la abcisa y la ordenada del punto en cuestión.

Si producimos un punto en x = 200, y = 10, mediante PLOT 200, 10 y, a continuación, lo queremos borrar, daremos la siguiente instrucción:

```
PLOT INVERSE 1 ; 200, 10
```

El comando PLOT se suele utilizar en instrucciones que fijan un punto en pantalla como origen o final de ciertos procesos o comandos y, en este sentido, estudiaremos, en las variables del sistema la forma de colocar este puntero donde se necesite sin hacer aparecer este punto en la pantalla.

EL COMANDO DRAW Y EL CALEIDOSCOPIO	FICHA 17

Efecto: Generar figuras circulares de contenido y color aleatorios.

Cómo conseguirlo: Dando al tercer parámetro del comando DRAW a, b, c, valores muy altos. Ejecute el siguiente programa y haga pruebas para obtener diversas formas, variando los parámetros de DRAW

```
10 INK RND * 2 : PLOT 80, 80 : DRAW 100, 10, 9000
20 INK RND * 2 : PLOT 110, 50 : DRAW 100, 10, 9000
30 PLOT 120, 100 : DRAW OVER 1; 100, 10, 700 : GOTO 30
```

Explicación: La sentencia DRAW con dos parámetros, dibuja arcos, calculando las coordenadas de gran cantidad de los puntos que los componen, uniendo éstos por medio de rectas que son apenas perceptibles, pero, al hacer el tercer parámetro muy grande y situar el punto inicial en las proximidades del centro de la pantalla son así de extraños.

IMPRESION EN PANTALLA CON DESPLAZAMIENTO VERTICAL	FICHA 18

Efecto: Desplazar una línea hacia arriba todo lo impreso en pantalla.

Cómo conseguirlo: Recurriendo a una rutina de la ROM, mediante esta instrucción:

```
RANDOMIZE USR 3582
```

Ejecute el programa de este ejemplo y haga pruebas para conseguir diferentes efectos.

```
 5 BORDER 0
10 RANDOMIZE USR 3582
15 PRINT
20 PRINT "Hola amigos:"
21 PRINT
25 PAUSE 50
30 PRINT "Buenas, buenas"
31 PRINT
35 PAUSE 50
40 PRINT "Aquí tele Spectrum"
41 PRINT
45 PAUSE 50
50 PRINT "poniéndose"
51 PRINT
55 PAUSE 50
60 PRINT "en comunicación directa"
61 PRINT
65 PAUSE 50
70 PRINT "con el gran mundo de"
```

```
 71 PRINT
 75 PAUSE 50
 80 PRINT "de los ´bitiosos´de"
 81 PRINT
 85 PAUSE 50
 90 PRINT "de la microinformática"
100 PRINT
1000 FOR X = 1 TO 17 : GO SUB 1010 : PAUSE 50 : NEXT X
1002 PRINT INVERSE 1; AT 13, 0;" ,, ADIOS
1003 PAUSE 0
1005 STOP
1010 RANDOMIZE USR 3582
1015 RETURN
```

BORDE "ELECTRICO"	FICHA 19 * "bordelec"

Efecto: Producir la sensación de una descarga eléctrica, mediante el cambio sucesivo y rápido del color del BORDE.

Cómo conseguirlo: Con la siguiente subrutina:

```
9990  FOR X=1 TO 20
9992  FOR Y=0 TO 6 : BORDER Y : PAUSE 1 : NEXT Y
9994  NEXT X
9996  RETURN
```

Haga la prueba de añadir esta subrutina al programa ejemplo de la ficha anterior y, además, sustituya la línea 1003 por esta:

```
1003  GOSUB 9990
```

Esta pequeña rutina combinada con algún tipo de "ruido" puede dar la sensación de un choque o descarga.

BLOQUEANDO PROGRAMAS QUE NO NECESITEN USAR INPUT	FICHA 20

Efecto: Impedir la copia del programa que esté en memoria, para lo cual éste no debe requerir la emisión o entrada de mensajes u órdenes.

Cómo conseguirlo: Ampliando la zona de impresión en pantalla a las 24 líneas de que dispone, aliminando las dos inferiores dedicadas a la comunicación usuario/máquina. Para ello se debe introducir la instrucción:

```
POKE 23659, 0
```

al comienzo del programa.

Añada la línea 9900 con esta instrucción al listado de la ficha anterior, sustituyendo el RETURN de la 9996 por un GOTO 9990.

Al ejecutar este nuevo listado, el BORDE "ELECTRICO" se producirá de una forma continua, así, cuando Ud. intente pararlo, se provocará la destrucción del programa.

Explicación: Al no existir zona de pantalla destinada a la impresión de mensaje, el computador entra en confusión, ya que en la dirección de memoria 23659 está guardado el valor que representa el número de líneas reservadas al computador, el cual, como consecuencia del POKE indicado, ha pasado a ser 0.

PROTEGIENDO UN PROGRAMA CONTRA STOP Y BREAK	FICHA 21

Efecto: Borrar el programa que actualmente esté en memoria o hacer que se "cuelgue" el Spectrum, al tratar de parar la ejecución del programa mediante STOP o BREAK por teclado.

Cómo conseguirlo: Alterando el contenido de la dirección de memoria 23613 o de la dirección 23614, antes de cada instrucción de GOTO, bucle FOR/NEXT y PRINT.

Pruebe esto sobre la rutina de la ficha del borde eléctrico, anadiendo ésta línea:

```
9990 POKE 23613,0
```

Ejecute esta rutina y, mientras se produce el BORDE "ELECTRICO", haga un STOP por teclado y espere.

Haga lo mismo pero cambiando la línea 9990, así:

```
9990 POKE 23614,0
```

Explicación: En estas direcciones está situada la variable del sistema ERR-SP que controla cualquier interrupción del programa que exija la emisión de un mensaje en pantalla. Si alteramos el contenido correcto de una o ambas, al computador entra en confusión y "rompe" el programa o se "cuelga".

ELIMINANDO EL SCROLL	FICHA 22

Efecto: Evitar las interrupciones de SCROLL? en las impresiones en pantalla, cuando superan las 22 líneas y bajo las condiciones que nos interesan.

Cómo conseguirlo: Manteniendo el contenido de la dirección de memoria 23692 por debajo de su valor normal que es 23, mediante, por ejemplo, un POKE 23692, 0.

Analice el resultado de ejecutar el siguiente programa:

```
10 FOR = 0 TO 100
20 IF X <> 55 THEN PRINT X : POKE 23692, 0 : GOTO 40
30 POKE 23692, 10 : FOR y = 0 TO 50 : PRINT "P" : NEXT y
40 NEXT X
```

Explicación: En la segunda instrucción de la línea 20, obligamos a que el contenido de la dirección en cuestión se mantenga a 0.

En la primera instrucción de la línea 30, colocamos el valor 10 en la dirección 23692, con lo cual el primer mensaje de SCROLL? aparecerá antes que si, por ejemplo, hubiéramos introducido el valor 18.

SALTANDO A UNA INSTRUCCION SITUADA EN UNA LINEA MULTISENTENCIA	FICHA 23

Efecto: Producir un salto incondicional a una instrucción situada dentro de una línea multisentencia.

Como sabemos, con GOTO podemos dirigir la lectura a cualquier línea de programa, con lo cual el programa sigue ejecutándose a partir de la primera instrucción situada en esa línea y no hay forma, en BASIC, de dirigirnos a una sentencia intermedia de forma directa, excepto la que se expone a continuación.

Cómo conseguirlo: Colocando el número de la línea que contiene la instrucción en cuestión en las direcciones de memoria 23618 y 23619, y, a continuación, introduciendo el número que corresponde a la instrucción, dentro de la línea anteriormente definida, en la dirección de memoria 23620.

Este ejemplo ayudará a fijar el proceso a seguir:

```
10 FOR X=0 TO 20
20 IF X=10 THEN POKE 23618,60 : POKE 23619,0 : POKE 23620,4
30 PRINT "no hay salto"
40 NEXT X
50 STOP
60 REM XXX : STOP : REM XXX : PRINT "hubo salto" : REM yyy :
   REM yyy
70 GOTO 40
```

Explicación: La variable del sistema NEWPPC, guarda la línea a la que hay que saltar y la NSPPC guarda la posición, dentro de ésa línea, que ocupa la sentencia a la que hay que saltar.

AHORRANDO MEMORIA CON VAL	Ficha 24

Efecto: Economizar el consumo de bytes a través de tratar los números como cadenas y utilizarlos, como tales, gracias la la función VAL.

Cómo conseguirlo: Allá donde necesitemos escribir un número, hagámoslo precedido de la función VAL y de acuerdo con la sintaxis que exige su uso.

Para ver esto con claridad y, además, poder comprobarlo, empecemos por cargar en memoria la rutina de la ficha "CONTROL DE MEMORIA", añadiendo al final de la línea 9994: – 400, en otras palabras, restar a lo *consumido en el programa*, lo que la propia rutina de control se lleva para, de esta forma, poder hacer comparaciones respecto a las pruebas que vamos a hacer a continuación.

Una vez hecho esto, escriba la línea:

```
10 PRINT 1
```

y corra el programa.

Podrá comprobar que en el epígrafe *consumido prog.* figuran 13 bytes, si sustituye, ahora, esta línea por

```
10 PRINT VAL "1"
```

verá que los bytes han bajado a 10.

¡Hemos ahorrado 3 bytes en una sola instrucción!

Explicación: Al usar números consumimos 6 bytes para el control de la coma flotante, mientras que el uso alternativo de VAL requiere 3 bytes, 1 por VAL y 2 por las comillas.

Las líneas multisentencia y la utilización de VAL es una forma sencilla y cómoda de ahorrar sistemática y cómodamente memoria.

AHORRANDO MEMORIA CON CODE	FICHA 25

Efecto: Economizar bytes mediante la transformación en números, por la función CODE, de los caracteres del Spectrum.

Sólo se puede usar entre el código del espacio (32) y el último de los caracteres definibles por el usuario (164).

Cómo conseguirlo: Igualando la variable al código del carácter que corresponda.

Supongamos que queremos igualar la variable X al número 127 y, si aún tenemos en memoria la rutina "CONTROL DE MEMORIA" modificada de acuerdo con las indicaciones dadas en la ficha anterior, podremos observar que consumimos 17 bytes al utilizar:

```
10 LET X = 127
```

Mientras que, al seguir el procedimiento aquí sugerido, ahorraremos 5 bytes.

Sustituya la línea 10 anterior, por esta:

```
10 LET X = CODE "©"
```

y compare.

Explicación: Es clara, si consideramos que sustituimos los tres caracteres del número 127 por uno sólo correspondiente al símbolo del *copyright*. Por lo demás vale el razonamiento dado para el ahorro de memoria con VAL.

AHORRANDO MEMORIA EN IF/THEN	FICHA 26

Efecto: Economizar bytes mediante la sustitución del operador de relación de la sentencia IF/THEN por el operador lógico NOT.

Este procedimiento puede ser utilizado, exclusivamente, cuando se comparen valores cuyos resultados posibles sea 0 y otro valor, positivo o negativo.

Cómo conseguirlo: Si en una línea de programa necesitamos comparar el contenido de la variable X, por ejemplo, con el valor 0, bien por igualdad como por desigualdad, sustituiremos esta relación por NOT X ó X.

En el siguiente programa:

```
10 INPUT X
20 IF X=0 THEN GOTO 10
```

podríamos sustituir la línea 20 por:

```
20 IF NOT X THEN GOTO 10
```

Sin que se alterase para nada el programa.

Si aún tiene el "CONTROL DE MEMORIA" compare el consumo de bytes en ambos casos, el cual alcanza los 7 bytes.

Dicho de otro modo NOT X equivale a X = 0 y X equivale a un valor de X distinto de 0.

Explicación: Se deriva de la tabla de la verdad del operador lógico NOT.

X	NOT X
1	0
0	1

USANDO LOS PARENTESIS LOGICOS	FICHA 27

Efecto: Resumir, en uno o varios paréntesis, las condiciones que se requieran, de tal forma que, si no se cumplen, el contenido total del paréntesis se ignora, en otro caso vale 1.

Cómo conseguirlo: La utilización de los paréntesis lógicos ofrece gran cantidad de posibilidades, consiguientemente, aquí se verán algunas formas para que el lector comprenda el proceso y lo maneje a su conveniencia.

Con el comando PRINT podemos usar los paréntesis lógicos con un considerable ahorro de memoria y, comodidad de programación.

El siguiente programa:

```
10 INPUT "Repetimos?. (s/n)" ; r$
20 PRINT ("De acuerdo, repetimos. "AND r$ = "s") + ("Vale, adios.
   "AND r$ = "n")
```

cumple la misma función que este otro:

```
10 INPUT "Repetimos?. (s/n)" . ; r$
20 IF r$ = "s" THEN PRINT "De acuerdo, repetimos."
30 IF r$ = "n" THEN PRINT "Vale, adios."
```

En el primer listado, y si ninguna de las condiciones impuestas en los dos paréntesis lógicos se cumple, la línea 20 sería equivalente a un PRINT sólo, ya que cada paréntesis valdría 0.

Podemos usar tantos paréntesis lógicos concatenados como queramos y, en cada uno de ellos, podemos imponer las condiciones que deseemos.

El programa cuyo listado se ofrece a continuación, permite mover una figura horizontalmente, a izquierda y derecha, según se apriete la tecla *A* o la *I*.

```
10 LET X = 11 : LET y = 11
20 PRINT AT x, y ; " * "
25 IF INKEY$ = " " THEN GOTO 15
30 IF INKEY$ = "A" THEN LET y = y – 1
40 IF INKEY$ = "I" THEN LET y = y + 1
50 IF y < = 1 THEN LET y = 1
60 IF y > = 29 THEN LET y = 29
70 GOTO 20
```

El mismo efecto que podríamos conseguir con este otro:

```
10 LET x = 11 : LET y = 11
20 PRINT AT x, y ; " * "
26 LET y = y – (INKEY$ = "A" AND y > = 1) + (INKEY$ = "I"
   AND y < = 29) : GOTO 20
```

Explicación: En la línea 26, cualquier paréntesis que se cumpla dará por resultado un 1, con lo cual a la variable "y" se le restará o sumará la unidad, si ninguno se cumple entonces la "y" permanece invariable.

Haga pruebas similares a los ejemplos anteriores con GOTO por ejemplo:

```
GOTO (500 AND = 1))
```

POKE p.e.:

```
POKE 23658, (8 ( x = 1 OR y = 5)), etc.
```

PROCEDIMIENTO RAPIDO PARA CARGAR G. D. U. **Gráficos definidos por el usuario**	FICHA 28

Efecto: Aumentar la velocidad de carga de los *bytes* correspondientes al diseño que exijan los nuevos gráficos definidos por el usuario en el programa que se vaya a ejecutar, y un considerable ahorro de memoria, tanto mayor cuanto más gráficos hubiera de definir.

Cómo conseguirlo: En el programa y la explicación que siguen, se desarrolla un ejemplo para dos G. D. U. pero, con pequeñas modificaciones, puede servir para cualquier número de gráficos.

```
 10 REM 0000000000000000
 20 RESTORE 510 : LET N = 1 : READ A$ : READ Y
 30 FOR X = 23818 TO 23833
 40 POKE USR A$ + X – Y, PEEK X
 50 LET N = N + 1 : IF N = 8 THEN READ A$ : READ Y
 60 NEXT X
 70 ::::::::::::::::::::::::::::
 80 ::::::::::::::::::::::::::::
 90 STOP:::::::::::::::::::::::
100 LET N = 1 : READ A$ : READ Y
110 FOR X = 23818 TO 23833
120 POKE X, PEEK (USR A$ + X – Y)
130 LET N = N + 1 : IF N = 8 THEN READ A$ : READ Y
140 NEXT X
150 ::::::::::::::::::::::::::::
160 ::::::::::::::::::::::::::::
170 ::::::::::::::::::::::::::::
180 ::::::::::::::::::::::::::::
190 STOP:::::::::::::::::::::::
200 LET N = 1 : READ A$
```

```
210 FOR X=0 TO 7
220 READ Z
230 POKE USR A$+X,Z
240 NEXT X
250 LET N=N+1 : IF N<=2 THEN READ A$ : GOTO 210
500 DATA "E",255,161,161,161,161,161,161,255,"F",255,161,
    161,161,255,161,161,255
510 DATA "E",23818,"F",23826
```

Explicación: Como se puede apreciar, gracias a las series de separadores, esta utilidad está dividida en tres partes.

La última, que comienza en la línea 200, es la primera que se debe utilizar. Mediante un RESTORE : GOTO 200, se configuran los nuevos caracteres gráficos por el procedimiento convencional. En este caso estamos trabajando sobre los caracteres *E* y *F* y según las series de números que figuran en el DATA de la línea 500.

En el subprograma que figura a partir de la línea 100 y hasta la 140, rellenamos los primeros 16 bytes útiles de la zona de memoria dedicada al BASIC –que previamente, en la línea 10, hemos reservado mediante dieciseis caracteres a continuación del REM, ceros en este ejemplo– con los *bytes* que configuran los nuevos caracteres gráficos y que han sido definidos en la etapa anterior.

Observe que, en el listado precedente, la dirección de comienzo útil de la zona dedicada al BASIC, es la 23818 y ello es debido a que, a la sazón, hay un Interface 1 conectado, si no fuera así, la dirección sería la 23760.

La razón de que sean estas direcciones las adecuadas y no otras, ya ha sido esbozada en otras fichas, no obstante, observamos que, la dirección del comienzo del BASIC viene dada por la variable del sistema PROG, cuyo contenido se determina mediante:

```
PRINT PEEK 23635 + 256 * PEEK 23636
```

con lo cuál obtendríamos, como dirección de arranque de esta zona, 23755, sin Interface 1 y 23813 con él conectado.

Si a cualquiera de estas dos direcciones le sumamos dos bytes para el control del número de línea, otros dos para la longitud de las misma y uno para la *palabra* REM, nos pondremos en 23759 ó 23817 –según el caso– lo cual nos indica que los caracteres (0 en este caso) a sustituir por los bytes que nos interesan comienzan en la si-

guiente dirección, 23818 ó 23760, según esté el Interface 1 conectado o no.

Una vez en este punto, borramos todas las líneas a partir de la 70 y hasta la 250 ambas incluidas, y ya estamos en condiciones de salvar el programa, usarlo por sí mismo o hacerlo formar parte de otro programa de más entidad, pero siempre al principio del mismo.

CREANDO LA LINEA 0	FICHA 29

Efecto: Convertir la primera línea de un programa en la línea 0, con el fin de que no pueda ser manipulada por los procedimientos usuales.

Cómo conseguirlo: Mediante la instrucción:

```
POKE 1 + PEEK 23635 + 256 * PEEK 23636, 0
```

Siguiendo este proceso:

1°.– Cargar el programa en cuestión en el Spectrum.

2°.– Introducir la instrucción citada en modo directo.

En este momento, la primera línea del programa (si el número es menor de 255) pasa a ser la línea 0 y ya no es posible editarla y, consiguientemente, modificarla o borrarla.

No obstante, claro está, si damos la instrucción contraria, y en el supuesto de que la primera línea del programa queremos que sea la 10, restituiremos el listado a las condiciones normales:

```
POKE 1 + PEEK 23635 + 256 * PEEK 23636, 10
```

Explicación: Por razones reiteradamente expuestas en fichas anteriores, la instrucción que damos en modo directo, altera el contenido de la segunda dirección de memoria de la zona dedicada al BASIC y que controla el byte menos significativo (números de línea comprendidos entre 1 y 255), con lo cual, al introducir un 0, el Spectrum ignora tal línea ya que la máquina sólo está preparada para manejar números de línea entre 1 y 9999.

Debemos observar que el byte menos significativo, en el caso de las direcciones de memoria que guarda la longitud de línea, corresponde al más significativo usualmente, de ahí el "uno" que figura al comienzo del POKE indicado.

CREANDO LINEAS MAYORES QUE 9999	FICHA 30

Efecto: Convertir los números de un grupo de líneas consecutivas en mayores que la máxima posible (9999) con el fin de hacerlas desaparecer del listado, pero manteniéndolas en el mismo.

También colocar una línea al final del programa para evitar su manipulación.

Cómo conseguirlo: Introduciendo en el ordenador, en primer lugar, las líneas que queremos hacer desaparecer, supongamos éstas:

```
500 REM "estas líneas"
510 REM "es una prueba"
520 REM "para ocultarlas"
```

a continuación damos la siguiente instrucción en modo directo:

```
POKE PEEK 23635 + 256 * PEEK 23636, 255
```

con lo cual el listado desaparece.

Si quisiéramos hacerlo reaparecer, tecleemos una línea de este tipo:

```
1 REM
```

una vez introducida esta línea, procedemos a borrarla por el sistema habitual y, a continuación y en modo directo:

```
POKE PEEK 23635 + 256 * PEEK 23636, 0
```

Con un LIST, veremos el listado nuevamente.

Explicación: Al hacer el *byte* más significativo, de los dos que controlan el número de la primera línea del programa, mayor que 9999, el Spectrum está imposibilitado para listar.

Lo dicho en esta ficha puede ayudarle si, al listar un programa, le aparecen unas pocas líneas en BASIC no consecuentes con la envergadura de aquél. Si quiere investigar lo que sucede, proceda como se ha explicado, borrando una a una las líneas que aparezcan y después las instrucción

```
POKE PEEK 23635 + 256 * PEEK 23636, 0
```

Otra posibilidad es pasar una línea, que contenga un mensaje de interés, al final del programa, de forma que sea prácticamente inviolable. Para ello, tecleemos la línea en cuestión en primer lugar, por ejemplo:

```
10 REM "© J. González"
```

y a continuación, en modo directo:

```
POKE PEEK 23635 + 256 * PEEK 23636, 40
```

Hecho esto, introduzcamos el resto del programa.

Como podrá observar, la línea 10 ha pasado a tener una numeración extraña que la hace practicamente intocable.

OCULTANDO INSTRUCCIONES Y MENSAJES CON POKE	FICHA 31

Efecto: Invisibilizar ciertas partes de un programa.

Cómo conseguirlo: Mediante los códigos de control de los colores en código máquina.

En la primera página de *El Juego de Caracteres* del Manual del Spectrum, aparecen los códigos de control de los atributos que pueden afectar a una posición de carácter. Así, si una determinada dirección de memoria contiene el código de control de la tinta (16) y el siguiente contiene un número correspondiente a uno de los colores (del 0 al 7), todos los caracteres que sigan a estas posiciones de memoria, irán afectadas de este color de tinta.

Veamos este ejemplo:

En la línea

```
10 REM ab
```

queremos hacer desaparecer los caracteres ab, para ello debemos empezar por introducir la línea anterior dejando dos espacios antes y después de estos caracteres, para poder introducir los oportunos códigos como veremos a continuación.

Ahora, en modo directo, y si no hay microdrives en servicio, damos las siguientes instrucciones:

```
POKE 23760, 16 : POKE 23761, 7 : POKE 23764, 16 ; POKE
23765, 0
```

Con lo cual desaparecen las letras en cuestión, en el supuesto de que el color de papel general sea blanco.

Explicación: Con los dos primeros POKES, introducimos en los *bytes* indicados –y previamente ocupados por los códigos de espacio que a tal efecto se dejaron– los códigos de control de tinta y el del color blanco.

En los dos bytes que siguen a los que ocupan los caracteres a ocultar, introducimos, gracias a los dos POKES siguientes, los códigos que restituyen a la tinta del color negro.

Es claro que cuantos más caracteres se traten de ocultar, más se tendrá que desplazar la dirección de los dos POKES últimos.

Lo expuesto aquí, puede ser de gran utilidad para hacer desaparecer la parte final de líneas multisentencia, con lo cual es difícil de preveer, por parte de un tercero, que exista nada.

DESFIGURANDO UN LISTADO EN BASIC	FICHA 32

Efecto: Convertir el listado de un programa en BASIC en su equivalente en bytes, sin que por ello deje de funcionar, y de forma que sea ininteligible.

Cómo conseguirlo: Salvando (SAVE) el programa en BASIC, no como tal, si no como los bytes que realmente lo componen.

Esto es fácil de hacer si se sigue el procedimiento que se indica a continuación.

Es de suponer que, cuando se pretende seguir un procedimiento como éste para impedir la copia del listado, es que intentamos protegerlo contra las miradas indiscretas, por tanto, en el ejemplo que nos servirá de guía en la explicación, utilizaremos los trucos ya estudiados que provocan la destrucción del programa cuando se intenta interrumpir su ejecución.

Admitamos que el listado a proteger corresponde a una rutina como esta:

```
10 REM listado
20 LET X=11 : LET y=11
30 PRINT AT, x, y ; "* ": POKE 23659, 0
40 LET y=y-(INKEY$="a" AND y>=1)+(INKEY$="1"
   AND y<=29) : GOTO 30
```

Con el POKE de la línea 30, obligamos a la destrucción del programa en el caso de que se pretenda detener la ejecución del programa.

Salvemos este programa mediante una instrucción que siga este modelo:

```
SAVE "Nombre" CODE 23500, 600 : RUN
```

El comando RUN al final de la instrucción, asegura la ejecución del programa después de la carga.

Para comprobar que el programa así salvado sigue funcionando correctamente, desconecte el Spectrum, y proceda a su carga mediante un LOAD " " CODE y observe que la cabecera del programa le da el mensaje:

bytes : *Nombre* y no, como es usual, **Program** : *Nombre*

Al final de la carga, el programa se autoejecuta, de tal forma, que ya no hay manera de ver el listado, puesto que un BREAK produciría la destrucción del programa.

Explicación: En primer lugar debemos saber que todos los programas que han sido salvados (SAVE) de alguna de las formas que provocan la autoejecución del programa al ser cargados (LOAD), necesitan salvar, no sólo el propio programa, sino también las variables del sistema actualizadas, las cuáles, según se puede ver en un mapa de memoria del Spectrum, comienzan en la dirección 23552.

Por otra parte, y como ya se ha expuesto de una u otra forma en fichas anteriores, un programa en BASIC es guardado, en forma de *bytes*, a partir de la dirección que viene dada por

```
PEEK 23635 + 256 * PEEK 23636
```

La cual nos dará, si no está el Interface 1 en servicio, 23755.

También debemos saber, como al respecto indica el manual del Spectrum, que para salvar *bytes*, debemos añadir a la sintaxis de SAVE la palabra CODE, situada en la tecla I, seguida de la dirección de memoria donde comienza el programa a salvar, una coma y, finalmente, la longitud del programa medida en *bytes*.

En función de todo esto, la razón por la cual se ha salvado el programa que nos interesa con un

```
SAVE "Nombre" CODE 23500, 600
```

es la siguiente.

"Nombre", evidentemente, corresponde al título que nos parezca conveniente darle al programa.

CODE es la palabra BASIC que ordena al computador que grabe o salve *bytes*.

23500, es un número redondo y fácil de memorizar, que corresponde a una dirección de memoria inferior a la de comienzo de las variables del sistema (23552) y que, por estar situada en el *buffer* de la impresora, no afecta para nada, a no ser claro está, que hubiéramos situado alguna información en esta zona.

600, es un número de bytes a salvar evidentemente superior al necesario.

Otra forma, rudimentaria pero eficaz, de desfigurar un listado es alterar el contenido de la variable del sistema CHARS que ocupa las direcciones de memoria 23606 y 23607, las cuales contienen, normalmente, 0 y 60.

Pruebe, con un programa en memoria, un POKE 23607, 0 y trate de pedir un listado.

Para restablecer la situación inicial haga un POKE 23607, 60.

La razón de que esto suceda así, es que CHARS guarda la dirección donde comienza el juego de caracteres que está en uso en el Spectrum y, al modificar este puntero, la máquina se dirige, a "leer" los caracteres, a un lugar equivocado.

LISTANDO UN PROGRAMA DESFIGURADO

FICHA 33

Efecto: Convertir en un listado BASIC normal un programa que, previamente, haya sido salvado en forma de *bytes*.

Cómo conseguirlo: Supongamos, en este caso, que queremos estudiar el listado del programa de la ficha anterior, el cuál fue protegido para evitar esto.

En primer lugar, vamos a proceder a su carga en el ordenador, pero cambiando la dirección de memoria a partir de la cual se van a situar los *bytes* del programa, por ejemplo:

```
LOAD " " CODE 29000
```

Esto quiere decir que, después de efectuada esta operación, el programa "Nombre" está situado a partir de la dirección 29000.

Ahora tendríamos que averiguar cuantos *bytes* han sido salvados por debajo de la dirección 23755 o dicho de otra forma, a nosotros sólo nos interesan los bytes del programa propiamente dicho y no los grabados para las variables del sistema.

Este tipo de investigación se esboza más tarde.

Supongamos que ya hemos averiguado que la grabación comenzó en la dirección 23500, de lo cual deducimos (23755-23500) que los primeros 255 bytes no son del listado.

En otras palabras, los bytes que nos interesan dan comienzo en la dirección 29255 y ocupan 355. De hecho, este último dato lo determinaríamos en el mismo proceso en el que se determina la dirección de comienzo.

Una vez en este punto, procederemos a salvar estos bytes con

```
SAVE "Nombre 1" CODE 29255,355
```

Con lo cuál habremos guardado exclusivamente en casete los bytes que nos interesan y que corresponden al programa anterior.

El próximo paso será generar, en la zona de memoria dedicada a contener el programa en BASIC, espacio suficiente como para que los bytes que tenemos en el casete bajo Nombre 1, encuentren alojamiento en este área, por sustitución del contenido de las direcciones a partir de la 23755.

Esto, por razones ya expuestas, lo podemos conseguir con una sóla línea que contenga una instrucción integrada por un REM seguido de tantos o más caracteres como *bytes* compongan "Nombre 1".

Teclee esta línea:

```
1 REM 000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
```

Introducida esta línea por el sistema que sea, procederemos a cargar el programa "Nombre 1", con la siguiente instrucción:

```
LOAD "  " CODE 23755
```

Terminado este proceso, bastará una instrucción de LIST para comprobar el resultado.

Explicación: Todas las necesarias se han dado a medida que el proceso anterior se desarrollaba, no obstante, la necesidad de introducir una línea REM seguida de caracteres antes de proceder a la última fase de carga, quizá no esté suficientemente clara.

La zona de memoria, dedicada a contener el programa en BASIC, empieza –si no hay microdrives en servicio–, en la dirección 23755 y además es dinámica, lo cual implica que, a partir de 23755, este área se irá dilatando a medida que un –y sólo un– programa BASIC lo requiera.

Dado que en nuestro caso vamos a pretender, posteriormente, introducir en la zona de programa BASIC y a partir de su comienzo, una serie de bytes, salvados como tales en casete, deberemos indefectiblemente "abrir" la zona de programa hasta la longitud que nos interese, mediante una instrucción BASIC y, dado que REM es la más apropiada por su inocuidad y flexibilidad, utilizamos la línea REM que se ha indicado en su momento.

FORMANDO UN GRAN REM EN LA LINEA 1	FICHA 34 * "gran REM"

Efecto: Aumentar el contenido de la instrucción REM de la línea 1 de una forma rápida y cómoda.

En muchas ocasiones, bien por que lo exija un determinado programa ensamblador, bien por que debemos transformar un programa BASIC camuflado en *bytes*, según se ha visto en la ficha anterior, o por cualquier otra razón, es imprescindible generar una línea 1 con un REM seguido de una gran cantidad de caracteres (normalmente usamos "ceros"), lo cual resulta tedioso y molesto, aún a pesar de la facilidad de autorepetición de las teclas del Spectrum, la cual, por otra parte, se hace más lenta a medida que el contenido de la instrucción aumenta.

Cómo conseguirlo: En base al listado que sigue y según el proceso que se explica a continuación.

```
 1 REM 00000
 2 REM 00000
 4 PRINT "Confirmar duplicación de línea 1. Si no, listar y duplicar.
   Si todo es conforme, apriete CONT y ENTER." : STOP
 5 LET prog = PEEK 23635 + 256 * PEEK 23636
 8 PRINT "Dir. inicio prog. BASIC : " ; prog
10 LET l1 = PEEK (prog + 2) + 256 * PEEK (prog + 3)
12 PRINT "long. línea 1 : " ; l1
30 LET l2 = prog + l1 + 6
35 PRINT "Dir. inic. cont. long. lin. 2 : " ; l2
40 LET l3 = PEEK 12 + 256 * PEEK (l2 + 1) + 4
45 PRINT "Long. lin. 2 : " ; l3
50 LET lt = l1 + l3
55 PRINT "Long. total nueva lin. 1 : " ; lt
```

```
 60 POKE (prog + 3), INT (lt/256)
 70 POKE (prog + 2), lt – PEEK (prog + 3) * 256
 80 POKE prog – 1 + l1 + 4, 42
 90 PRINT : PRINT "Caracteres disponibles en LA ACTUAL línea 1 :
    " ; lt – 2
100 PRINT : PRINT "SI SON SUFICIENTES APRIETE CONT Y EN-
    TER"
110 PRINT : PRINT "SI NO, DUPLIQUE LA LINEA 1 EN LA DOS Y
    EJECUTE EL PROGRAMA OTRA VEZ"
120 STOP
130 FOR i = prog + 5 TO 64512 : IF PEEK i <> 13 THEN POKE
    i, 48 : NEXT i
140 LIST
```

Una vez introducido el anterior programa en el Spectrum, colocamos, a continuación del REM de la línea 1, una serie de "ceros", cinco en el ejemplo.

Una vez hecho esto, editamos la línea 1 y le cambiamos su número por el 2, de tal forma que nos encontraremos con dos líneas iguales, la 1 y la 2, según se ve en el anterior listado.

El siguiente paso es ejecutar el programa, con lo cuál la pantalla nos mostrará un mensaje que pretende alertarnos de la posibilidad de haber olvidado duplicar la línea 1 en la 2, ya que de ser así, destruiríamos el programa.

Una vez superada esta interrupción, se producirá otra para preguntarnos si los caracteres que, tras el proceso, tiene la línea 1 son suficientes, y, de no serlo, volveríamos a editar la línea 1, duplicándola, ejecutando el programa nuevamente, hasta que se considere que el REM de la línea 1 contiene los caracteres necesarios. En ese momento, continuaríamos –según muestran las instrucciones– con la ejecución del programa, para lograr la línea 1 con tantos "ceros" como caracteres disponibles haya en el nuevo REM. Llegado este momento, todas las líneas pueden ser borradas excepto la primera y después, con un MERGE si es necesario incorporar otro programa o proceder, en definitiva, como exija el motivo de toda esta manipulación.

Explicación: Ante todo, el lector interesado en esta explicación, podrá reducir considerablemente la extensión del programa a partir del momento en que comprenda el proceso que maneja.

Debemos recordar, una vez más, que la zona de memoria donde se

almacena un programa BASIC en el Spectrum, está definida por la variable del sistema PROG, contenida en las direcciones 23635 y 23636, y esto es lo primero que averiguamos en la línea 5, igualando este valor a la variable PROG.

Como sabemos, los dos primeros *bytes* a partir del que contenga PROG están dedicados al número de línea y los dos siguientes contienen la longitud de dicha línea –medido en caracteres y considerando el comando BASIC como un carácter– incluyendo el REM y el ENTER del fin de línea.

Y esto es lo que fijamos en la línea 10, dando a la variable *l1* este valor.

En la línea 30, buscamos la dirección de memoria que contiene el primero de los dos bytes que controlan la longitud de la línea dos, la cuál vendrá dada por la siguiente suma:

prog	dirección comienzo del BASIC
2	bytes para el número de la línea 1
2	bytes para control de la longitud de la línea
l1	longitud de la línea 1, excepto los cuatro bytes anteriores
2	bytes para el número de la línea 2
l2	primera dirección de memoria dedicada al control de la longitud de la línea 2

Razón por la cual igualamos la variable *l2* a la expresión prog + l1 + 6.

En la línea 40, y en función de lo expuesto anteriormente, averiguamos la longitud de la línea 2, más los 4 *bytes* de control, haciéndola igual a la variable l3.

En la 50, y mediante una simple suma, determinamos cual será la longitud total que tomará la nueva línea 1, ya que *l1* contiene la longitud de la actual línea 1, y *l3*, según acabamos de ver, la longitud de la actual línea 2.

En las líneas 60 y 70, cambiamos el contenido de las direcciones que controlan la longitud de la línea 1, con los valores correspondientes a la nueva línea 1, que vandrá dada por la variable *lt* definida en la línea 50.

La línea 80, introduce el código ASCII del asterisco en la dirección de memoria que ocuparía el código del ENTER –fin de línea - de la primera línea, para orientar al lector en el proceso seguido hasta aquí.

En la línea 90, se ordena la impresión de los caracteres disponibles en la nueva línea 1, que serán los indicados por *lt* menos los dos *bytes* correspondientes a REM y ENTER.

Finalmente, y cuando el proceso se da por concluido, pasamos a la línea 130, donde se produce la inserción de los códigos ASCII correspondientes al "cero" en todas las direcciones de memoria que siguen a REM (prog + 5) y hasta que una de ellas contenga el código del fin de línea (ENTER).

EVITANDO QUE UN PROGRAMA SE AUTOEJECUTE

FICHA 35

Efecto: Impedir que un programa se ejecute una vez cargado, con lo cual se podrá inspeccionar el listado.

Cómo conseguirlo: Utilizando el comando MERGE, en lugar de LOAD. Pruebe esta posibilidad sobre el programa del ejemplo de las dos fichas anteriores.

Impidiendo la autoejecución de un programa, estamos en condiciones de listarlo y, consiguientemente, proceder a eliminar las instrucciones destinadas a destruir el programa en caso de BREAK o STOP y, en definitva, de poder estudiar aquellos aspectos que nos interesen.

Los programas en código máquina no se pueden parar si no se introduce alguna rutina previa que provoque una interrupción, en este sentido se puede utilizar algún programa de los que existen en el mercado, con este tipo de utilidad encontrará, además, dónde comienza el código máquina y su longitud.

Una vez conseguidos estos objetivos, procederíamos, como en casos anteriores, a mover el código máquina a otras posiciones de memoria y así poder actuar en el sentido que convenga.

La inspección de rutina en código máquina, requiere de un desensamblador o, de otra forma, se puede averiguar el contenido de las posiciones que interese con un programa de este tipo:

```
10 FOR X=29000 TO 31000 : PRINT X, PEEK X : NEXT
```

En el supuesto de que el campo de investigación esté situado entre estas dos direcciones de memoria.

LLAMANDO A UNA RUTINA EN CODIGO MAQUINA	FICHA 36

Efecto: Ejecutar un programa escrito en código máquina.

Cómo conseguirlo: Por la configuración del Spectrum, sólo podemos dirigir la lectura a una rutina en código máquina partiendo de unos tipos específicos de instrucciones, cada una de las cuales producen la ejecución de la rutina, pero a su vez cumplen distintas funciones. Si en modo directo o en una línea de programa damos la instrucción

```
RANDOMIZE USR 29000
```

estamos provocando la ejecución de una subrutina en código máquina situada a partir de la línea 29000.

En el caso de usar:

```
LET X = USR 29000
```

Provocamos, igualmente, la ejecución de la rutina, pero, además, estamos valorando la variable X con el contenido del par de registros *bc* en el momento de producirse el *ret*orno de la subrutina en cuestión. Este valor de X puede ser usado, como el número que es, en la forma que convenga.

Un caso similar al anterior se consigue con

```
PRINT USR 29000
```

Pero, en este caso, obtendremos la impresión del contenido del par de registros *bc*.

INVESTIGANDO UN LISTADO	FICHA 37

Efecto: Conseguir un listado completo del programa que está en memoria, eliminando las instrucciones de autoprotección sin que ello altere la ejecución del programa.

Cómo conseguirlo: En función de todo lo expuesto en todas las fichas anteriores, podemos fijar el siguiente criterio.

1°.– Si el programa se autoejecuta, lo impediremos por el método más adecuado de entre los expuestos en otras fichas.

2°.– Coloquemos un BORDE que contraste con el color de PAPEL de la pantalla.

3°.– Listar el programa. Si tiene impresora, un LLIST le podrá ayudar mucho ya que los caracteres ocultos en pantalla, salen impresos en papel.

4°.– Edite las líneas demasiado cortas o aquéllas otras que estén sucedidas de una línea en blanco y desplace el cursor para ver si este se detiene en algún lugar, esto es indicativo de que hay códigos de control. Proceda a borrarlos con DELETE.

5°.– Un poco más pesado será localizar las variables que hayan podido ser introducidas en modo directo, siendo nuestro único recurso localizarlas a través de sucesivas pruebas del tipo PRINT A, PRINT A$, PRINT B, etc.

No obstante con la rutina CONTROL DE MEMORIA, podemos determinar el espacio ocupado por las variables en memoria antes y después de un RUN, ya que este comando borra las variables, y, si estas fueron introducidas en modo directo, habrán desaparecido.

6°.– Colocar las líneas mayores que 9999, si existen, en los primeros lugares del listado, de acuerdo con lo visto.

7°.– Si el programa está transformado en *bytes*, debemos seguir el proceso indicado para convertir el listado en BASIC.

8°.– Si hay algún tipo de información guardada en el REM de la primera línea, aténgase a lo dicho en la ficha 28.

9°.– En el caso de que aparezca en algún momento de la investigación una rutina en código máquina, deberíamos determinar en primer lugar si existen cadenas de caracteres, para lo cual buscaríamos, con un bucle apropiado, los códigos ASCII transformados en sus caracteres equivalentes y, a continuación, recurriríamos a un desensamblador. INFRARED es un desensamblador sencillo y eficaz.

10°.– Ya dentro de una lista de nemónicos deberemos trocearla en tantas partes como *ret* existan.

Todo el trabajo de investigación de un programa debe ir precedido de un estudio previo del mismo, por medio de una utilidad del tipo TRANS EXPRESS, SPY o COPION.

Cuando aplique sus energías en el sentido que está orientada esta ficha, sentirá una mayor satisfacción si lo emprende con el ánimo de perfeccionar sus conocimientos.

EMISION DE MENSAJES EN FORMA DE TELETIPO	FICHA 38 *"teleprint"

Efecto: Provocar la impresión en pantalla de forma automática, carácter a carácter y, de izquierda a derecha, de cadenas de caracteres.

Cómo conseguirlo: Mediante una rutina similar a ésta:

```
10  LET ti=2 : LET td=12 : LET E$=CHR$ 13
20  BORDER 1 : PAPER 6 : INK 1 : CLS
30  PRINT AT 21,0 ; "■" ; AT 21,31 ; "■" ; AT 0,31; "■" AT 0,0 ;
    "■"
40  READ s$
50  IF s$="@" THEN PAUSE 50 : GOTO 20
60  IF s$="&" THEN STOP
70  PRINT TAB ti;
80  FOR X=1 TO LEN s$
90  IF s$(X)=" " THEN IF PEEK 23688<td THEN PRINT '
100 BEEP .06, CODE s$(X)-65 : PRINT s$(X) ;
110 IF s$(X)=E$ THEN PRINT TAB ti;
120 NEXT X
130 GOTO 40
9000 DATA "En un lugar de la Mancha", E$; "de cuyo nombre no quie-
     ro acordarme", E$, "Vivía no ha mucho, " @"
9010 DATA "un hidalgo de los de lanza en astillero" , E$, "rocín flaco
     y galgo corredor", "&"
```

Una vez entendido el mecanismo de impresión de caracteres, y en general de la rutina, el lector podrá conseguir sus propios efectos.

La combinación de "teleprint" con el SCROLL de pantalla hacia arriba, puede dar resultados satisfactorios.

Explicación: En la línea 10, fijamos la tabulación izquierda (ti), a la derecha (td) y fijamos una variable alfanumérica (E$) a un carácter que no provoca ningún tipo de impresión (CHR$ 13 = ENTER).

En la línea 20 y 30, se crean unos efectos para configurar el marco dentro del cuál aparecen los textos que se generan posteriormente.

En el READ de la línea 40, se leen las cadenas que, sucesivamente, se irán imprimiendo.

La línea 50, contiene una condición que obliga a dirigir la lectura del programa a la línea 20, si la cadena leida por READ es igual al símbolo @, esto implica el borrado de la pantalla y la generación de un marco nuevo. El efecto conseguido, en definitiva, es como si una nueva página comenzara.

En la línea 60, la condición está impuesta en función de que la cadena leida sea igual o no al carácter &, con lo cual el programa interrumpirá la ejecución o continuará. Es claro que, si el programa fuera de mayores dimensiones y "teleprint" fuera sólo una parte del mismo, la sentencia STOP estaría sustituida por un GOTO a una línea que permitiera la continuación del mismo.

Una vez superadas las dos condiciones anteriores y en la línea 70, obligamos a que la cadena leida en 40 comience la impresión a partir del margen tabulado por *ti* en la próxima línea de pantalla disponible. El punto y coma que cierra la instrucción, determina que el siguiente carácter de la cadena en cuestión se imprima justo a continuación.

Entre las línea 80 y 120, está el bucle fundamental de la rutina que comienza por establecer el número de veces que se repite y que queda entre 1 (primer carácter de la cadena actual) y LEN S$ (longitud de la cadena actual).

La doble condición lineal dictada en la línea 90, impone, en primer lugar, que, si el carácter de la cadena que a la sazón se esté manipulando –S$ (X)–, es un espacio, entonces se investigue si la posición de impresión es menor que la tabulación derecha (td).

Si tal ocurre, la impresión comienza en una nueva línea, si no continua en la misma.

Lo que se pretende con estos condicionantes es evitar romper una palabra, para ello comprobamos que el carácter sea –o no– un espacio y, después, su posición en la pantalla, gracias a la variable del sistema S POSN (23688) que controla la siguiente posición del carácter a imprimir.

En la línea 100, imprimimos en todo caso, el carácter que el bucle esté manejando, tras emitir un sonido de duración, fijado aquí en 0.06, y de tono variable de acuerdo con el código del carácter en cuestión.

Antes de salir del bucle, línea 110, se comprueba si el carácter en cuestión ha sido igual a E$ (código de ENTER), si fue así será indicativo de que el siguiente carácter corresponderá a una nueva cadena, con lo cual comenzará una nueva línea de impresión tabulada a *ti*.

En 120 se cierra el bucle y en 130 se obliga a dirigir la lectura del programa a la línea que leerá la sigueinte cadena a imprimir. Después están las series de DATAS que abastecen al READ de la línae 40 y que, en definitiva, configuran el mensaje a emitir en forma de teletipo.

ALMACENANDO CADENAS EN FORMA DE BYTES. SU MANEJO	FICHA 39 * STOCK ASCII

Efecto: Ahorrar memoria al guardar información que deba ser procesada.

Cómo conseguirlo: Transfiriendo, carácter a carácter, todas las cadenas a almacenar a unas posiciones de memoria consecutivas a partir de una dirección de memoria determinada por el programador.

La rutina que sirve al efecto, puede seguir este modelo:

```
9990 INPUT "dirección inicial ?" ; di : CLEAR (di – 1)
9991 INPUT "Repita! . Dirección inical ?" di : LET n = 1 : LET d = di
9992 PRINT "cadena" ; n : INPUT "tecleela" ; s$
9993 POKE d, LEN s$ + 1
9994 FOR X = 1 TO LEN s$ : POKE d + X, CODE s$ (X) : NEXT X
9995 LET n = n + 1 : LET d = d + LEN s$ + 1 : PRINT AT 0,0 ; "Ca-
     dena " ; s$; " transferida." ; AT 2,0; "bytes ocupados : " ; d – di
9996 INPUT "Otra cadena? (s/n) " ; r$ ; IF r$ = "n" OR r$ = "N" THEN
     STOP
9997 CLS : GOTO 9992
```

Al ejecutar este programa, y por mediación de él, se introducen las cadenas a almacenar de acuerdo con las hipótesis establecidas anteriormente.

Ahora bien, la manipulación de las mismas, dependerá del objetivo que persiga el programador, pero, en todo caso, deberá considerarse que el byte anterior al primer carácter de cada cadena contiene la propia longitud de la cadena, medida en bytes –o caracteres–, más uno.

Las posiciones de memoria que siguen a este, y hasta la longitud de la cadena, la ocuparán los sucesivos códigos ASCII de los caracteres que la componene.

Estas consideraciones, permiten el manejo de este almacén con toda libertad.

No obstante, y a título de ejmplo, el siguiente programa permite ver la forma de localizar una cadena cualquiera dentro del almacén.

Ejemplo:

```
10 INPUT "Número de la cadena?" ; n
20 INPUT "dirección inicial del almacén?" ; di : LET d = di
30 FOR X = 1 TO n – 1 : LET d = d + PEEK d : NEXT X
40 FOR X = d + 1 TO  d + PEEK d – 1 : PRINT CHR$ PEEK X ; :
   NEXT X
50 GOTO 10
```

Ejecute la primera rutina y, a través de ella, introduzca una serie de cadenas a partir de la dirección de memoria, por ejemplo, 30000 y, a continuación, ejecute la segunda y responda a las preguntas que le plantee.

Haga sus comprobaciones y pruebas.

Explicación: La idea consiste en almacenar los códigos ASCII de los sucesivos caracteres de una cadena, a partir de una dirección dada, para ello se ha seguido este criterio:

En la línea 9990, se pide la dirección de memoria a partir de la cual queremos generar el almacén de códigos ASCII y, a continuación, variamos la posición de RAMTOP a una posición de memoria menos de donde va a comenzar el almacenamiento. Este CLEAR provocará, a su vez, el borrado de la variable *di*.

En la línea 9991, volvemos a pedir la dirección inicial (di), por la razón ya expuesta, y fijamos los valores iniciales de la variable *n*, con la que controlaremos el número de cadenas que se vayan introduciendo, y la variable *d*, que irá llevando la cuenta de los bytes que se ocupen.

La línea 9992, simplemente, solicitará la próxima cadena a almacenar.

El POKE de la línea 9993 es fundamental, ya que guarda, en la dirección de memoria dada por *d*, el número de posiciones de memoria que ocupa la cadena a almacenar más un byte, que corresponde al primero de todas las cadenas y que encierra la longitud dada por el byte de control más los que requiere la cadena –uno por carácter–.

En el bucle situado en la línea 9994, se produce el almacenamiento de los caracteres de la cadena a almacenar.

En 9995, se actualizan las variables de control y se ordena la impresión de la situación después de la transferencia de la cadena al almacén ASCII.

Las líneas restantes, 9996 y 9997, permiten reiniciar –o no– el ciclo.

Con respecto a la rutina de manipulación propuesta a título de ejemplo, poco se puede añadir respecto a lo que se deduce de los comentarios anteriores. En todo caso, en el bucle de la línea 30 se determina la dirección de comienzo de la cadena cuya posición en el almacén se solicita en la línea 10 y, en el bucle de la línea 40, reconfiguramos e imprimimos la cadena en cuestión.

ALMACENAMIENTO DE IMAGENES EN RAM. TRANSFERENCIA DE IMAGENES ENTRE PANTALLA Y RAM	FICHA 40 * "transfer"

Efecto: Almacenar, en forma de *bytes*, cualquier imagen que contenga el fichero de imágenes (pantalla) en una zona de memoria que interese, para poder disponer de ella en todo momento.

Cómo conseguirlo: Tranfiriendo, byte a byte, la imagen que deseamos, y que, actualmente contenga la pantalla, a una zona de memoria RAM cuya dirección de comienzo debemos indicar y guardar.

Esto se logra gracias, fundamentalmente, a dos rutinas en código máquina –CM. TRANS0 y CM. TRANS7– y, secundariamente, a través del programa BASIC que sigue:

```
  5 INPUT "Dir. Inicial? " ; DO : CLEAR DO – 1
 10 LOAD * "m" ; 1 ; CM.TRANSØ " CODE : LOAD * "m" ; 1 ;
    "CM. TRANS7" CODE
 20 INPUT "A RAM = 1.A FI = 0" ; I0
 30 IF I0 = 1 THEN GOTO 150
 40 CLS : GOTO 120
 50 INPUT "Columna?" ; c : POKE 30000, c
 60 INPUT "Línea ? " ; c : POKE 30001, c
 70 INPUT "Ancho? " ; c : POKE 30002, c
 80 INPUT "Alto? " ; c : POKE 30003, c
 90 INPUT "Dir.Inicial? " ; DO
100 POKE 30004, DO – 256*INT (DO/256) : POKE 30005, INT (DO/
    256)
110 RETURN
120 GO SUB 50
130 RANDOMIZE USR 59200
140 PRINT # 0; AT, 0, 0; "Transferida! . UNA TECLA" : PAUSE 0:
    GOTO 20
```

```
150  GO SUB 50
160  RANDOMIZE USR 59000
170  INPUT "Salvar? (s/n) " ; r$
180  IF r$ <> "s" THEN GOTO 20
190  INPUT "Nombre imagen? " ; n$ : LET D1 = PEEK 30002*PEEK
     30003*8 ; SAVE n$ CODE D0, D1
200  PRINT # 0; AT 0, 0; "Dir. Inic = " ; D0 + D1; " UNA TECLA" :
     PAUSE 0
210  GOTO 20
```

Una vez que Ud. sepa el manejo del programa anterior, podrá, sin dificultad, usarlo para almacenar una serie de imágenes en una zona de memoria y "mostrarlas", en el momento que le convenga, en la pantalla.

Explicación: En la línea 5, se pide la dirección de memoria a partir de la cual comienza el banco de imágenes que se piensa transferir a RAM o, que de hecho, ya contiene la memoria RAM. A continuación se establece una nueva RAMTOP justo en la dirección adecuada.

La línea 10, contiene dos instrucciones de carga de las rutinas en código máquina procedentes de microdrives. En el caso de que Ud. quiera cargar del casete que, opcionalmente acompaña al libro, deberá cambiarlas por:

```
LOAD "CM.TRANS0" CODE : LOAD "CM.TRANS7" CODE
```

A continuación se adjunta un listado de ambas rutinas en código máquina para completar esta ficha.

Con la nueva RAMTOP colocada en una dirección de memoria menos que aquélla a partir de la cual guardaremos, tanto los bytes de las imágenes como de las rutinas "CM.TRANS0" y "CM.TRANS7" y, con estas en memoria, pedimos, por mediación de la 20, el tipo de transferencia: al almacén RAM (1) o a la pantalla (0).

Con esta información en la variable I0, se toma la decisión de dirigir la lectura del programa a la línea 150 –si la transferencia es a RAM– o a la línea 120 –si la transferencia es el fichero de imágenes–.

En ambas líneas (120 y 150), se obliga a recorrer la subrutina que se inicia en la línea 50 y finaliza con el RETURN de la 110.

En esta subrutina, se fijan los parámetros que servirán para la transferencia de cada imagen por medio de las oportunas rutinas en código máquina y que son

Columna: determina la columna en la pantalla donde comienza –o queremos que comience, según el tipo de tranferencia de de que se trate– la imagen a transferir.

Por exigencia de las rutinas en código máquina, este dato se coloca en la dirección de memoria 30000.

Línea: determina la línea de pantalla donde comienza –o queremos que comience, según el tipo de transferencia de que se trate– la imagen a transferir.

Ancho: se refiere al ancho, medido en caracteres, que ocupa –o ocupará al ser transferida– en pantalla, la imagen que nos interese.

Alto: se refiere al alto, medido en caracteres, que ocupa –o ocupará al ser transferida– en pantalla, la imagen que nos interese.

En la línea 90, se pregunta por la dirección de memoria inicial (D0) del lugar de la RAM adonde queremos transferir la imagen o, en otro caso, el lugar donde comienza, en la RAM, la imagen a transferir a pantalla.

En la línea 100, colocamos el valor anterior (D0) en las direcciones 30004 y 30005, por necesidades de las rutinas en código máquina.

La línea 110, devuelve el control del programa a la línea siguiente de donde saliera el GOSUB correspondiente.

En las 130 y 160, se hace la llamada a la rutina que corresponda, según el tipo de transferencia de que se trate.

La línea 140, es evidente y la línea 170 y sucesivas no requieren otro comentario que el relativo al valor D0 + D1, el cual es absolutamente imprescindible anotar, tanto para poder almacenar la siguiente imagen, si de esto se trata, como para poder disponer de ella para transferirla a la pantalla.

Los listatos que se ofrecen a continuación, corresponden a "CM. TRANS0" y "CM.TRANS7", los cuales, por otra parte, están grabados en el casete que se puede adquirir con este manual.

En cualquier caso, los listados completan esta ficha, cuya utilidad será muy alta para aquellas personas que trabajen con gráficos en general o gusten de decorar sus pantallas de textos.

CM.TRANS7

Dirección de comienzo: 59000

Longitud: 110 bytes.

59000	221	59045	0
59001	33	59046	72
59002	48	59047	195
59003	117	59048	173
59004	221	59049	230
59005	94	59050	33
59006	0	59051	0
59007	221	59052	80
59008	86	59053	107
59009	1	59054	175
59010	221	59055	30
59011	78	59056	32
59012	2	59057	6
59013	221	59058	8
59014	70	59059	203
59015	3	59060	59
59016	221	59061	210
59017	110	59062	185
59018	4	59063	230
59019	221	59064	130
59020	102	59065	203
59021	5	59066	34
59022	245	59067	16
59023	197	59068	246
59024	213	59069	95
59025	229	59070	25
59026	62	59071	84
59027	7	59072	93
59028	186	59073	6
59029	218	59074	8
59030	158	59075	126
59031	230	59076	225
59032	33	59077	119
59033	0	59078	13
59034	64	59079	175
59035	195	59080	185
59036	173	59081	202
59037	230	59082	211
59038	62	59083	230
59039	15	59084	35
59040	186	59085	28
59041	218	59086	26
59042	170	59087	119
59043	230	59088	195
59044	33	59089	198

59090	230	59102	230
59091	221	59103	209
59092	78	59104	193
59093	2	59105	241
59094	35	59106	20
59095	20	59107	16
59096	123	59108	169
59097	145	59109	201
59098	60	59110	0
59099	95		
59100	26		
59101	16		

CM.TRANS0

Dirección de comienzo: 59200

Longitud: 110 bytes

59200	221	59234	64
59201	33	59235	195
59202	48	59236	117
59203	117	59237	231
59204	221	59238	62
59205	94	59239	15
59206	0	59240	186
59207	221	59241	218
59208	86	59242	114
59209	1	59243	231
59210	221	59244	33
59211	78	59245	0
59212	2	59246	72
59213	221	59247	195
59214	70	59248	117
59215	3	59249	231
59216	221	59250	33
59217	110	59251	0
59218	4	59252	80
59219	221	59253	107
59220	102	59254	175
59221	5	59255	30
59222	245	59256	32
59223	197	59257	6
59224	213	59258	8
59225	229	59259	203
59226	62	59260	59
59227	7	59261	210
59228	186	59262	129
59229	218	59263	231
59230	102	59264	130
59231	231	59265	203
59232	33	59266	34
59233	0	59267	16

59268	246	59289	142
59269	95	59290	231
59270	25	59291	221
59271	84	59292	78
59272	93	59293	2
59273	6	59294	35
59274	8	59295	20
59275	225	59296	123
59276	126	59297	145
59277	18	59298	60
59278	13	59299	95
59279	175	59300	126
59280	185	59301	16
59281	202	59302	230
59282	155	59303	209
59283	231	59304	193
59284	35	39305	241
59285	28	59306	20
59286	126	59307	16
59287	18	59308	169
59288	195	59309	201

TRANSFORMANDO UN NUMERO DECIMAL EN SU EQUIVALENTE EN OTRA BASE	FICHA 41 * "dec-b"

Efecto: Pasar un número en base 10, a otra base.

Cómo conseguirlo: Mediante la siguiente rutina:

```
9990 INPUT "Número decimal? " ; n: LET n1 = n : INPUT "Nueva base? " ; b : LET n$ = " "
9991 LET d = INT (n – b*INT (n/b)) : LET o$ = CHR$ (d + 48 + (7AND d > 9)): LET n$ = o$ : LET n = INT (n/b)
9992 IF n = 0 THEN PRINT n1, n$ : STOP
9993 GOTO 9991
```

El programa pide el número decimal a transformar y, la nueva base, respondiendo el número de ambas bases.

Explicación: En la línea 9990, se solicitan los datos anteriores y se fija el valor del número en base 10 en dos variables: en n se recoge el número dado por teclado, con lo cuál trabajaremos en el resto del programa, y en n1 se guarda el valor inicial, con el fin de no perderlo. También se crea la variable de caracteres n$, que irá recogiendo, en forma de caracteres, los sucesivos dígitos que se generen en el proceso de conversión.

En la primera instrucción de la línea 9991, se calculan los dígitos, de izquierda a derecha, que conformarán el nuevo número. En la segunda instrucción, se transforma el dígito en su código ASCII correspondiente, en este sentido, debemos observar que el paréntesis lógico da por resultado la suma –o no– de 7 si el dígito calculado es mayor que nueve, con lo cual entramos en los códigos alfabéticos.

Dentro de la misma línea, y a continuación, se va rellenando la variable de caracteres n$, de izquierda a derecha, gracias al orden de yuxtaposición de las variables alfanuméricas. Finalmente se actualiza la variable *n* para el siguiente proceso.

En 9992, si el valor actual de n es cero, se ordena la impresión del valor inicial del número decimal (n1) y la de la cadena que contiene los caracteres de los dígitos del número equivalente en la nueva base.

En caso contrario, el GOTO de la línea 9993 reinicia el ciclo.

TRANSFORMANDO UN NUMERO DE UNA BASE CUALQUIERA A DECIMAL	FICHA 42 * "b-dec"

Efecto: Pasar un número en una base dada a su equivalente en base 10.

Cómo conseguirlo: Mediante la siguiente rutina

```
9990 POKE 23658,8 : INPUT "Base actual?" ;b : INPUT "Número?";
     n$ : LET n1 = 0
9991 FOR I = LEN n$ TO 1 STEP – 1
9992 LET n = CODE n$ (1) –48– (7AND n$ (1) > = "A")
9993 LET e = LEN n$ – 1 : LET n1 = n1 + n*b ↑ e : NEXT I
9994 PRINT n$, n1
```

El programa pide la base actual y el número en cuestión, y retorna el número anterior y el correspondiente en base 10.

Explicación: En la primera instrucción de la primera línea, obligamos a la utilización de mayúscula, manejando la dirección de memoria indicada, según vimos en la ficha correspondiente.

En el resto de la línea, se definen las variables que necesitamos para el proceso.

De la línea 9991 hasta la 9993, se establece un bucle que se recorrerá, en sentido inverso, tantas veces como caracteres contenga la cadena (n$) que representa al número en la base dada.

En 9992, se calcula el equivalente decimal del dígito del número a transformar y que actualmente esté estudiando el bucle.

En 9993, se aplica exactamente el Teorema Fundamental de la Numeración.

SELECCIONANDO INFORMACION DE UN BANCO DE DATOS	FICHA 43 * "Select"

Efecto: Localizar palabras que comiencen por un grupo de letras –o claves– determinadas.

Cómo conseguirlo: La rutina que se expone a continuación, selecciona la información en base a unas claves, pero, si la organización del banco de datos lo permite, y con pequeñas modificaciones en el listado, podríamos generalizarla para cualquier tipo de localización. En todo caso, lo que aquí se pretende es hacer que el lector interesado en este tema conozca, al menos, una forma de organización de la cual partir para ulteriores desarrollos.

La parte inicial del listado, situada antes de las tres líneas de separadores, genera, simplemente, un almacenamiento de información en el *array* d$, con el fin de que la rutina propiamente dicha, sea capaz de buscar en algún banco de datos.

```
 5  DIM c$ (1, 3) : DIM d$ (100, 5)
 7  FOR Z = 1 TO 100 : LET d$ (Z) = CHR$ (Z + 32) : NEXT Z
 8  LET d$ (5) = "*abc" : LET d$ (80) = "# qwer" : LET d$ (50) =
    "@ poiu"
 9  : : : : : : : : : : : : : : : : : : : : : : : : : : : : :
    : : : : : : : : : : : : : : : : : : : : : : : : : : : : :
    : : : : : : : : : : : : : : : : : : : : : : : : : : : : :
10  STOP : LET n = 0 : INPUT "Clave? " ; c$ (1)
20  FOR X = 1 TO 100
40  FOR y = 1 TO 3
50  LET X$ = d$ (X) (y TO y)
60  IF X$ = c$ (1) (1 TO 1) THEN LET y = 3 : LET n = n + 1 : PRINT
    d$ (X) ' : NEXT Y
70  IF X$ = c$ (1) (2 TO 2) THEN LET y = 3 : LET n = n + 1 : PRINT
```

```
    d$ (X) ´: NEXT y
 80 IF X$ = c$ (1) (3 TO 3) THEN LET y = 3 : LET n = n + 1 : PRINT
    d$ (X) ´: NEXT y
 90 NEXT y
100 NEXT X
110 PRINT ´X ; "datos contrastados con " ´n ; "seleccionados."
```

Explicación: La explicación que sigue, parte del supuesto de admitir que, la información almacenada en el array d$, ha sido guardada según el criterio de colocar a cada dato un asterisco al comienzo de cierto tipo de información (sellos españoles), un símbolo @ al principio de otro tipo (sellos italianos) y otro distinto para otras (sellos portugueses), de tal forma que, si deseamos obtener por ejemplo, relación de los sellos italianos que tenemos, deberemos responder a la pregunta "clave?", con un @ , o, para conocer que sellos españoles o italianos tenemos deberíamos dar la clave * @ .

Es claro que, en la línea 7 del listado actual no hay sellos si no, simplemente, un carácter cualquiera y, en la 8, hemos introducido alguna información con una clave, a título de ejemplo.

La rutina propiamente dicha, comienza en la línea 10, donde se inicializa la variable n, que nos servirá de contador de los items encontrados, y se pide la clave para la localización de los mismos.

Con el bucle que se abre en la línea 20 y se cierra en la 100, establecemos la condición necesaria para hacer tantas comparaciones como datos contenga el array que contenga la información –d$ en nuestro caso–.

En la línea 40, generamos la condición para comparar los tres primeros caracteres de cada dato, con los símbolos de la clave dada en la línea 10.

En la 50, seleccionamos un carácter –(y TO y)– del dato de d$ que, a la sazón, se esté manipulando, en función de la variable X del bucle de la línea 20.

En las líneas 50, 60 y 70, se realizan, efectivamente las comparaciones, se actualizan los valores de las variables afectadas y se ordena la impresión del dato contrastado y seleccionado.

Las tres últimas líneas no requieren explicación.

En el mismo orden de cosas, las claves podrían ser utilizadas como un elemento seleccionador de información a transferir a otro array, del cual, a su vez, se podría solicitar datos aún más precisos sobre un mismo tema, o, por el contrario, se pueden establecer prioridad de claves que, mediante una concatenación de condiciones, obtengan detalles puntuales sobre ciertos aspectos del contenido de la base de datos.

En la página 173 y siguientes del Manual del Spectrum, encontrará la totalidad de estas variables, algunas de las cuales hemos tratado detalladamente en anteriores fichas.

KSTATE –23552– 23559

Estos bytes mantienen el teclado bajo el control del sistema, el cual cheque la situación del mismo con una frecuencia de 50 veces cada segundo. Los cuatro bytes finales guardan la siguiente información respecto a la primera tecla apretada:

23556.–El código ASCII del carácter de la tecla apretada, en mayúscula. Si el teclado no registra ninguna tecla pulsada, este byte contiene IIIIIIII (255).

23557.– La velocidad con que se ha de repetir la impresión en pantalla del carácter de la tecla si se mantiene apretado. Este parámetro lo toma de la variable REPPER (23562).

23558.–El tiempo que debe mantenerse apretada una tecla para que entre en autorepetición.

Este parámetro lo toma de la variable REPDEL (23561).

23559.–El código ASCII del carácter de la tecla apretada, en minúscula.

Las cuatro primeras posiciones tienen las mismas funciones, si una segunda tecla es apretada y la primera retorna a su situación normal.

Con respecto al teclado, se puede añadir que, la instrucción *in a, (254)* permite enviar una señal al microprocesador, procedente del

teclado. Esta señal se produce considerando el teclado dividido en ocho semifilas de cinco teclas consecutivas cada una.

Apretando cualquiera de las teclas de una de estas semifilas, se deja uno de los bits, del byte de la señal, a 0. En el caso de que no haya ninguna tecla pulsada, todos los bits son 1, como ya se dijo.

LAST K – 23560

El *byte* guardado en esta dirección de memoria, representa el código ASCII de la última tecla apretada.

REDPEL Y REPPER

Ya han sido comentadas en la variable anterior. Sólo añadir que los valores que contienen al inicializar el sistema pueden ser alterados, introduciendo en sus direcciones de memoria los valores adecuados.

CHARS – 23606 y 23607

Estas dos direcciones de memoria –más 256–, guardan al inicializar el sistema, los *bytes* que determinan la dirección de memoria donde comienza el juego de caracteres estándar. Estos bytes añadidos corresponden a los 32 primeros códigos ASCII que, como se puede comprobar en la página 183 del Manual del Spectrum, no corresponden estrictamente a caracteres representables en pantalla.

En el caso de que el usuario quiera utilizar su propio juego de caracteres, deberá cambiar el contenido de los dos bytes de CHARS con la dirección donde comience el nuevo *set*.

MODE – 23617

Esta variable puede tener cierto interés, para aquellos programas que requieran trabajar en modo gráfico, para ello bastaría con introducir, con un POKE, en la dirección de memoria que indica esta variable, el valor 2, con lo cual el cursor cambia a G.

Si se quiere cambiar al modo extendido, el número sería 1.

Con 0 volvemos el cursor al modo L.

BORDCR – 23624

Esta variable es interesante para dotar a la zona baja de la pantalla –la dedicada a los mensajes del computador– de sus propios atributos. Pruebe el efecto que producen estos POKES:

```
POKE 23624, 120
POKE 23624, 130
```

E PPC – 23625 y 23626

Estas dos direcciones de memoria, guardan el número de línea donde está situado el puntero que la señala en pantalla.

Una aplicación derivada de la manipulación de estos bytes, consiste en introducir en estas posiciones, al principio de un programa de listado largo, los valores que correspondan al número de línea donde queremos situar el puntero, así, al ejecutar el programa, y producir una interrupción, tendremos el puntero situado en la línea deseada, siempre que apretemos la tecla ENTER exclusivamente –sin LIST– para provocar el listado en pantalla, o, de otra forma y trás el STOP –pedir la edición de la línea actual, que será la deseada, con lo cual, al pulsar ENTER para introducirla en el listado nuevamente, tendremos en pantalla la sección del listado que nos interese.

FRAMES – 23672, 23673 y 23674

Gracias a estas tres direcciones de memoria, que controlan el número de "imágenes" que se envían, por segundo, a la pantalla y que son 50, podemos confeccionar un reloj digital, un contador de segundos, un cronómetro, etc.

UDG – 23675 y 23676

Los bytes que contienen estas dos direcciones de memoria, determinan el comienzo de la zona dedicada a los gráficos definidos por el usuario.

Si alteramos esta dirección haciéndola más alta obtendremos un cierto espacio para introducir bytes de nuestro interés.

Haga la prueba, por ejemplo, de cambiar el contenido de 23675 con 100 y tendrá a su disposición una codificación especial, y si introduce 255 los caracteres se habrán redefinido simulando nubes de puntos.

COORDS – 23677 y 23678

La primera dirección guarda el número de la abscisa (coordenada X) del último punto dibujado.

La segunda dirección guarda la ordenada (coordenada y) de este mismo punto. Apoyándonos en esta variable del sistema podemos imponer condiciones de borde a la impresión de puntos en pantalla.

SPOSN – 23688 y 23689

Algo similar a lo dicho en COORDS, podría ser aplicado a esta variable, pero referido a la posición de impresión de un carácter en la pantalla.

EPILOGO

Al llegar a este epígrafe –final del libro–, Ud. habrá podido comprobar que he evitado, en la medida de los posible, los largos listados que suelen ser tediosos y a nada conducen, excepto consumir papel.

En mi opinión, a partir de cierto nivel de formación presumible del lector, la información que debe contener el libro debe ser puntual, concisa y clara.

Si lo he conseguido a lo largo de estas páginas, tendré mi primer motivo de satisfacción.

En cuanto a las fichas y su contenido, me he visto obligado a seguir un criterio de selección muy estricto, con el fin de tratar de mantenerlas dentro de una tónica de interés y utilidad similar para el lector al que me dirijo.

Si no le he defraudado, habré logrado mi principal objetivo.